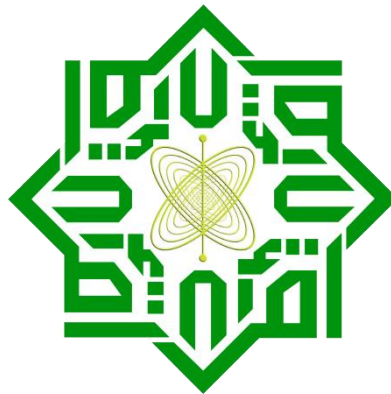


PENERAPAN *OBJECT ORIENTED PROGRAMING* PADA PEMBUATAN PROGRAM RENTAL MOBIL

Laporan Hasil *Project* Aplikasi

Diajukan Sebagai salah Satu Syarat Untuk Menyelesaikan Ujian Tengah Semester
Pada Mata Kuliah Pemrograman Lanjut.



UIN SUSKA RIAU

Oleh:

| | |
|--------------------------------|--------------------|
| DAFFA IKHWAN NURFAUZAN | 12250110979 |
| FAJRI | 12250110382 |
| MUHAMMAD ADITYA RINALDI | 12250111048 |

**JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI SULTAN SYARIF
KASIM RIAU PEKANBARU**

2023

KATA PENGANTAR

Assalamu'alaikum Warahmatullahi Wabarakatuh.

Alhamdulillah Rabbi'alam, tidak henti-hentinya penulis ucapkan puji dan syukur kehadiran Allah SAW yang telah melimpahkan kepada penulis nikmat kesehatan, kesempatan, ilmu pengetahuan, sehingga penulis dapat menyelesaikan laporan hasil project aplikasi yang berjudul Penerapan Object Oriented Programming pada Pembuatan Program Rental Mobil. Sholawat dan salam juga penulis sampaikan kepada Nabi dan Rasul yang telah mengajarkan islam sehingga sampai kepada penulis sebagai pedoman kehidupan.

Laporan hasil project aplikasi ini dibuat untuk memenuhi tugas mata kuliah Pemrograman Lanjut, penulis menyadari bahwa dalam penulisan makalah ini masih banyak kekurangan, Oleh karena itu, apabila dikemudian hari terdapat kekeliruan maka akan dilakukan perbaikan sebagaimana mestinya. Akhirnya, penulis berharap semoga laporan ini dapat memberikan perubahan positif pada pelaksanaan tugas mata kuliah Pemrograman Lanjut pada jurusan Teknik Informatika UIN Suska Riau.

Pekanbaru, 22 Mei 2023

Penulis

DAFTAR ISI

| | |
|----------------------------------------------------|-----|
| KATA PENGANTAR | ii |
| DAFTAR ISI | iii |
| DAFTAR GAMBAR | v |
| DAFTAR TABEL | vi |
| BAB 1 PENDAHULUAN | 1 |
| 1.1 Latar Belakang | 1 |
| 1.2 Rumusan Masalah | 3 |
| 1.4 Tujuan | 3 |
| 1.3 Manfaat Penelitian | 4 |
| BAB 2 KAJIAN PUSTAKA | 5 |
| 2.1 Inheritance | 5 |
| 2.1.1 Pengertian Inheritance | 5 |
| 2.1.2 Dasar-Dasar Inheritance | 5 |
| 2.1.3 Aturan Inheritance | 6 |
| 2.1.4 Pengaruh Control Akses pada Pewarisan | 6 |
| 2.2 Method Overriding dan Method Overloading | 7 |
| 2.3 Abstract Class | 8 |
| 2.3.1 Pengertian Abstract Class | 8 |
| 2.3.2 Aturan Abstract Class | 8 |
| 2.3.3 Keuntungan Method Abstract | 9 |
| 2.4 Polymorphism | 10 |
| 2.4.1 Pengertian Polymorphism | 10 |
| 2.4.2 Polimorphism Statis | 11 |

| | |
|--------------------------------------------|----|
| 2.4.3 Polimorphism Dinamis..... | 11 |
| 2.5 Interface | 11 |
| 2.5.1 Pengertian Interface | 12 |
| 2.5.2 Cara Mendeklarasikan Interface | 12 |
| 2.5.3 Implementasi Interface | 12 |
| 2.5.4 Pewarisan Pada Interface | 13 |
| 2.6 Encapsulation | 13 |
| 2.6.1 Pengertian Encapsulation | 14 |
| 2.6.2 Control Akses | 14 |
| 2.6.3 Tanpa Access Specifier | 15 |
| BAB 3 PEMBAHASAN | 16 |
| 3.1 Tampilan Aplikasi..... | 16 |
| 3.2 Pembahasan Kode Program..... | 20 |
| BAB 4 PENUTUP | 28 |
| 4.1 Kesimpulan..... | 28 |
| 4. 2 Saran | 29 |
| DAFTAR PUSTAKA | 30 |

DAFTAR GAMBAR

| | |
|----------------------------------------|----|
| Gambar 1 Tampilan Menu Awal..... | 16 |
| Gambar 2 Tampilan Pemilihan Mobil..... | 16 |
| Gambar 3 Detail Mobil..... | 17 |
| Gambar 4 Pemilihan Nomor Polisi..... | 17 |
| Gambar 5 Pengisian Data Diri..... | 18 |
| Gambar 6 Peringatan Isi Data..... | 18 |
| Gambar 7 Contoh Pengisian Data..... | 19 |
| Gambar 8 Pemesanan Berhasil..... | 19 |
| Gambar 9 Flowchart Inheritance..... | 20 |
| Gambar 10 Membuka Aplikasi..... | 21 |
| Gambar 11 Kode Menu Awal | 22 |
| Gambar 12 Kode Menu Awal 2..... | 22 |
| Gambar 13 Pilihan Mobil 1..... | 23 |
| Gambar 14 Pilihan Mobil 2..... | 23 |
| Gambar 15 Kendaraan 1..... | 24 |
| Gambar 16 Kendaraan 2..... | 24 |
| Gambar 17 Kode Detail Mobil..... | 25 |
| Gambar 18 Kode Detail Mobil 2..... | 25 |
| Gambar 19 Kode Template Mobil..... | 26 |
| Gambar 20 Kode Template Mobil 2..... | 26 |
| Gambar 21 Kode Konfirmasi..... | 27 |
| Gambar 22 Kode Konfirmasi 2..... | 27 |

DAFTAR TABEL

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Dalam era perkembangan teknologi informasi saat ini, penerapan konsep Pemrograman Berorientasi Objek (OOP) telah menjadi sangat penting dalam pengembangan perangkat lunak. OOP adalah paradigma pemrograman yang berfokus pada objek sebagai unit utama dalam pengorganisasian dan pemodelan sistem. Konsep ini memungkinkan pengembangan perangkat lunak untuk mengorganisir kode menjadi objek-objek yang saling berinteraksi, memfasilitasi pengembangan yang lebih terstruktur, modular, dan dapat digunakan kembali.

Salah satu industri yang mengandalkan sistem perangkat lunak yang efisien dan terorganisir adalah industri rental mobil. Bisnis rental mobil memerlukan sistem yang dapat mengelola penyewaan mobil secara efisien, melacak inventaris mobil, menghitung biaya sewa, serta memfasilitasi proses reservasi dan pengembalian mobil. Dalam konteks ini, penerapan konsep OOP dapat memberikan manfaat yang signifikan.

Menurut Grady Booch (2007), mengatakan “OOP memungkinkan untuk mengorganisir sistem yang kompleks menjadi terstruktur, dengan objek sebagai elemen dasar yang saling berinteraksi. Dalam proyek rental mobil, OOP dapat memberikan fleksibilitas dan keterbacaan yang tinggi dalam mengelola mobil, pelanggan dan transaksi.”

Dalam proyek rental mobil, penerapan OOP dapat memungkinkan pengembang untuk mengorganisir kode menjadi objek-objek yang merepresentasikan entitas bisnis seperti mobil, pelanggan, transaksi dan lain sebagainya. Setiap objek memiliki atribut yang merepresentasikan karakteristik

dan metode yang merepresentasikan perilaku atau operasi yang dapat dilakukan pada objek tersebut.

Dengan menggunakan OOP, pengembang dapat membuat kelas-kelas yang mewakili entitas bisnis tersebut dan menggabungkannya menjadi sebuah sistem yang terintegrasi. Ini memungkinkan pengembang untuk dengan mudah menambahkan fitur baru, memperbaiki kesalahan dan memperluas fungsionalitas sistem tanpa mempengaruhi bagian lainnya. Selain itu, kode yang menggunakan konsep OOP juga dapat digunakan kembali dalam proyek-proyek lain, menghemat waktu dan usaha dalam pengembangan perangkat lunak.

Menurut Erich Gamma , “Design pattern dalam OOP adalah alat yang kuat untuk mengatasi masalah umum dalam pengembangan perangkat lunak. Dalam proyek rental mobil, penggunaan design pattern seperti singleton untuk manajemen akses database atau strategi untuk perhitungan biaya sewa dapat meningkatkan fleksibilitas dan keterbacaan kode.”

Dengan melihat kompleksitas bisnis rental mobil dan kebutuhan akan sistem yang efisien dan terorganisir, penerapan konsep OOP dalam proyek rental mobil menjadi penting. Dalam makalah ini, penulis akan menjelaskan tentang penerapan OOP dalam pengembangan sistem rental mobil, meliputi desain kelas, hubungan antar kelas, penggunaan pewarisan, abstrak, polimorfisme, enkapsulasi, *overloading*, *overriding* dan manfaat lainnya yang diberikan oleh OOP.

Penulis akan membahas secara rinci penerapan Program Berorientasi Objek dalam proyek rental mobil sebagai solusi yang efektif dalam mengoperasional bisnis. Penulis akan menjelaskan konsep – konsep Program Berorientasi Objek (OOP) yang relevan dan bagaimana mereka diterapkan dalam proyek rental mobil yang akan dibuat. Penulis juga akan menulis contoh implementasi nyata yang menggambarkan manfaat dan tantangan dalam penerapan OOP dalam proyek rental mobil.

1.2 Rumusan Masalah

Dalam laporan ini, terdapat beberapa rumusan masalah yang berkaitan dengan Penerapan OOP pada Project Rental Mobil, rumusan masalahnya yaitu :

1. Bagaimana cara menerapkan Program Berorientasi Objek (OOP) pada *project* rental mobil?
2. Bagaimana penerapan *Inheritance* pada OOP dalam pembuatan *project* rental mobil?
3. Bagaimana penerapan Method *Overloading* dan Method *Overriding* pada OOP dalam pembuatan *project* rental mobil?
4. Bagaimana penerapan *Abstract class* dan *Polymorphism* pada OOP dalam pembuatan *project* rental mobil?
5. Bagaimana penerapan *Interface* dan *Encapsulation* pada OOP dalam pembuatan *project* rental mobil?
6. Bagaimana penerapan percabangan(*If Else*) dan perulangan(*Looping*) pada OOP dalam pembuatan *project* rental mobil?

1.3 Batasan Masalah

Pada laporan ini, penulis hanya berfokus pada pengerjaan penerapan Program Berorientasi Objek (OOP) pada pembuatan *project* aplikasi rental mobil pada Bahasa Java.

1.4 Tujuan

Tujuan dari laporan ini yaitu :

1. Untuk mengetahui bagaimana cara menerapkan Program Berorientasi Objek (OOP) pada *project* rental mobil
- 1.3 Untuk mengetahui cara menerapkan *inheritance* pada OOP dalam pembuatan *project* rental mobil
2. Untuk mengetahui cara penerapan Method *Overloading* dan Method *Overriding* dalam pembuatan *project* rental mobil

3. Untuk mengetahui cara penerapan *Abstract Class* dan *Polymorphism* pada OOP dalam pembuatan project rental mobil
4. Untuk mengetahui cara penerapan *Interface* dan *Encapsulation* pada OOP dalam *project* rental mobil
5. Untuk mengetahui cara penerapan percabangan(*If Else*) dan perulangan(*Looping*) pada OOP dalam project rental mobil

1.3 Manfaat Penelitian

Manfaat laporan ini adalah untuk mengetahui bagaimana cara menerapkan OOP pada *project* rental mobil, dan juga untuk menambah wawasan dan ilmu pengetahuan terkait bagaimana cara penerapan Program Berorientasi Objek (OOP) dalam penggunaan sehari-hari. Laporan ini juga bertujuan untuk mendapat nilai Ujian Tengah Semester (UTS) pada mata kuliah Pemrograman Lanjut.

BAB 2

KAJIAN PUSTAKA

2.1 Inheritance

Program Berorientasi Objek yang pertama yaitu *Inheritance*. Pada subbab ini berisi penjelasan mengenai pengertian *inheritance*, dasar-dasar *inheritance*, aturan *inheritance*, dan pengaruh Control Akses pada Pewarisan.

2.1.1 Pengertian Inheritance

Inheritance dalam pemrograman merupakan konsep yang memungkinkan pembuatan baru (kelas turunan) yang mewarisi sifat dan perilaku dari kelas yang sudah ada (kelas induk atau super class). Dengan menggunakan inheritance, kelas turunan dapat menggunakan dan memperluas fungsi-fungsi yang sudah didefinisikan dalam kelas induk, sehingga memungkinkan untuk membagi dan mengatur kode dengan lebih efisien.

Dalam inheritance, kelas induk menyediakan kerangka dasar yang umum untuk kelas – kelas turunan. Kelas turunan dapat menambah atribut dan metode tambahan, serta mengganti atau memperluas perilaku yang diwarisi dari kelas induk. Hal ini memungkinkan untuk membangun hierarki kelas yang lebih spesifik, dimana kelas turunan memiliki karakteristik lebih khusus daripada kelas induk.

2.1.2 Dasar-Dasar Inheritance

1. *Inheritance* (Pewarisan) adalah sebuah konsep pemrograman dimana sebuah class dapat menurunkan atau mewariskan dapat berupa atribut dan method yang dimilikinya atau dimiliki *class* kepada *class-class* lain.
2. *Class* yang akan diturunkan disebut sebagai kelas induk (*parent class*), super class atau base class, sedangkan class yang menerima warisan atau

ahli waris dapat disebut sebagai kelas anak (*child class*), *sub class* atau *derived class*.

3. Untuk melakukan *inheritance* atau pewarisan pada sebuah *class* di dalam java menggunakan keyword “*extends*”.
4. *Inheritance* mempermudah dalam membentuk suatu pola pewarisan antar *class* sehingga memungkinkan programmer untuk membuat suatu *class* atau ahli waris yang sama dan efisien.

2.1.3 Aturan Inheritance

1. *Subclass* hanya dapat memiliki satu *superclass* (tidak boleh ada banyak pewaris).
2. *Konstruktor subclass* harus memanggil *konstruktor superclass* dengan kata kunci “*super()*” atau “*super(parameter)*”.
3. *Subclass* dapat mengubah nilai bidang atau metode yang diwarisi dari *superclass* dengan menyimpannya.
4. Penentu akses (*public*, *private* dan *protected*) dapat digunakan untuk menentukan akses ke bidang atau metode *superclass* yang diwarisi oleh *subclass*.
5. Ketika bidang atau metode *superclass* dinyatakan *final*, subclass tidak dapat mengubah nilai bidang tersebut atau mengganti metode tersebut.
6. Subclass dapat menambahkan bidang atau metode baru, tetapi tidak dapat menghapus bidang atau metode yang diwarisi dari *superclass*.

2.1.4 Pengaruh Control Akses pada Pewarisan

Beberapa jenis control akses pada pewarisan

1. Pengubah akses publik (*Public*)
 - a. Bidang atau metode yang dideklarasikan secara *public* dapat diakses oleh *subclass*, kelas lain atau bahkan dari luar *package*.

- b. Hal ini memungkinkan *subclass* atau kelas lain yang menggunakan warisan untuk mengakses *field* atau metode yang diperlukan dari *superclass*.
- 2. Pengubah akses pribadi (Private)
 - a. *Field* atau metode yang dideklarasikan secara sebagai *private* tidak dapat digunakan oleh *subclass* atau *class* lain, meskipun mereka menggunakan pewarisan. Hal ini memastikan bahwa *field* atau *method* yang digunakan hanya digunakan oleh *superclass* dan tidak dapat dimodifikasi oleh *subclass* atau *class* lain.
- 3. Pengubah akses terlindungi (Protected)
 - a. *Field* atau *method* yang dideklarasikan sebagai *protected* dapat digunakan oleh *subclass* dan *class* lain yang berada dalam package yang sama dengan *superclass*.
 - b. Hal ini memungkinkan *subclass* untuk mengakses *field* atau metode dari *superclass* yang diperlukan untuk pewarisan.

2.2 Method Overriding dan Method Overloading

- 1. *Method overriding* adalah salah satu fitur penting dalam *inheritance* (pewarisan) di Java, yang memungkinkan *subclass* untuk mengganti implementasi dari sebuah *method* (yang diwarisi dari *superclass*). Dengan *method overriding*, *subclass* dapat menambahkan atau mengubah perilaku dari *method* yang diwarisi sesuai dengan kebutuhan *subclass*.
- 2. *Method overloading* adalah fitur pada Java *inheritance* yang memungkinkan untuk memiliki beberapa *method* dengan nama yang sama memungkinkan untuk memiliki beberapa *method* dengan nama yang sama di dalam sebuah *class*, namun dengan parameter yang berbeda. Hal ini sangat berguna dalam *inheritance* (pewarisan) karena *subclass* dapat menggunakan *method overloading* untuk menambahkan implementasi dari *method* yang diwarisi dari *superclass*.

2.3 Abstract Class

Program Berorientasi Objek yang berikutnya yaitu *abstract class* dan *polymorphism*. Pada subbab ini berisi penjelasan mengenai pengertian *abstract class*, aturan *abstract class*, keuntungan *method abstract*, pengertian *polymorphism* dan pembagiannya.

2.3.1 Pengertian Abstract Class

Abstract class dalam pemrograman adalah sebuah *class* yang tidak dapat diinstansiasi secara langsung, tetapi berfungsi sebagai kerangka dasar untuk kelas-kelas turunannya. *Abstract class* sering digunakan untuk menyediakan struktur, perilaku umum dan metode *abstract* yang diwaris dan diimplementasikan oleh kelas-kelas turunan.

2.3.2 Aturan Abstract Class

1. Keyword “*abstract*”, kelas *abstract* harus dideklarasikan dengan menggunakan keyword “*abstract*”. Dengan demikian, compiler akan mengetahui bahwa kelas tersebut merupakan kelas *abstract* dan tidak dapat di – *instantiate*
2. Tidak dapat di-*instantiate*, karena kelas *abstract* tidak dapat di-*instantiate*, anda tidak dapat membuat objek langsung dari kelas tersebut. Namun kelas *abstract* dapat digunakan sebagai superclass untuk membuat subclass yang mewarisi sifat dan perilaku dari kelas tersebut.
3. Memiliki setidaknya satu *method abstract*. Setiap kelas *abstract* harus memiliki setidaknya satu *method abstract*, yaitu *method* yang hanya dideklarasikan(tanpa diimplementasikan) dan harus diimplementasikan pada subclass yang mewarisi kelas *abstract* tersebut.
4. Dapat memiliki *method non-abstract*, selain memiliki *method abstract*, kelas *abstract* juga dapat memiliki *method non-abstract* (biasa disebut

method concrete method). Method non-abstract merupakan metode yang diimplementasikan dan dapat langsung digunakan oleh subclass.

5. Tidak dapat digunakan secara langsung, karena kelas abstract tidak dapat diinstansiasi, kelas tersebut juga tidak dapat digunakan secara langsung dalam sebuah program. Namun, kelas abstract dapat diwarisi oleh subclass yang dapat digunakan dalam program.

2.3.3 Keuntungan *Method Abstract*

Penggunaan method abstract pada pemrograman berorientas objek memiliki beberapa keuntungan, di antaranya:

1. Mengurangi duplikasi kode. Dengan menggunakan *method abstract*, kita dapat menentukan perilaku atau aksi yang samaa pada beberapa kelas yang berbeda. Dalam hal ini, kita tidak perlu menulis ulang kode yang sama setiap kelas, sehingga mengurangi duplikasi kode yang dapat menyebabkan kesalahan dan mengurangi efisiensi dalam pengembangan program.
2. Mempermudah pemeliharaan program. Dalam kasus perubahan perilaku atau aksi pada program, kita hanya perlu mengubah *method abstract* pada *superclass*, dan semua *subclass* yang mewarisi method tersebut akan otomatis terpengaruh oleh perubahan tersebut. Dengan demikian, perubahan kode hanya perlu dilakukan pada satu tempat, sehingga mempermudah pemeliharaan program.
3. Meningkatkan fleksibilitas program. Dengan menggunakan *method abstract*, kita dapat membuat kelas-kelas yang dapat sesuaikan dengan kebutuhan program, dan memperbolehkan *subclass* untuk mengimplementasikan perilaku kelas *abstract* sesuai dengan kebutuhan mereka. Hal ini meningkatkan fleksibilitas program dan memungkinkan kita untuk menyesuaikan perilaku kelas sesuai dengan kondisi yang berbeda.

4. Meningkatkan *Polimorfisme*. *Polimorfisme* adalah kemampuan sebuah objek untuk menerima banyak bentuk atau tipe. Dengan menggunakan method *abstract*, kita dapat membuat kelas-kelas yang berbeda namun memiliki perilaku yang sama. Dalam hal ini, kita dapat menggunakan objek kelas *abstract* untuk merepresentasikan semua objek dari kelas-kelas yang mewarisi kelas *abstract* tersebut, dan melakukan operasi pada objek tersebut tanpa perlu mengetahui jenis objek yang sesungguhnya.

2.4 Polymorphism

Polymorphism adalah konsep dalam pemrograman berorientasi objek di mana objek dari kelas yang berbeda dapat dianggap sebagai objek dari kelas yang sama. Lebih tepatnya, polymorphism memungkinkan suatu objek untuk mengambil bentuk (tipe) yang berbeda ketika digunakan dalam konteks yang berbeda. Dengan kata lain, objek yang sama dapat menunjukkan perilaku yang berbeda tergantung pada kelas yang digunakan untuk mengaksesnya.

2.4.1 Pengertian Polymorphism

Polimorfisme terbagi menjadi dua suku kata yaitu, *Poly* yang berarti banyak dan *Morfisme* yang berarti bentuk. Dalam ilmu sains, polimorfisme (*Polymorphism*) adalah sebuah prinsip dalam biologi dimana organisme atau spesies memiliki banyak bentuk serta tahapan (*stages*). Prinsip tersebut diterapkan juga pada bahasa Java. Polimerfisme dalam OOP merupakan sebuah konsep OOP dimana class memiliki banyak “bentuk” method yang berbeda, polimorfisme (*Polymorphism*) adalah kemampuan suatu objek untuk mengambil banyak bentuk atau tipe. Dalam pemograman berorientasi objek, polimorfisme terjadi ketika suatu objek dapat digunakan sebagai objek dari kelas yang berbeda-beda, meskipun objek tersebut tersebut sebenarnya adalah instance dari kelas yang sama atau subclass dari kelas tersebut, meskipun namanya sama. Maksud dari “bentuk” adalah isinya yang berbeda, namun tipe data dan parameternya berbeda.

2.4.2 Polimorphism Statis

Polimorfisme statis (*static polymorphism*) adalah jenis polimorfisme dimana metode yang akan dipanggil ditentukan pada saat kompilasi, bukan pada saat runtime. Konsep ini juga dikenal juga dikenal dengan nama *early binding* atau *compile-time binding*.

Polimorfisme statis biasanya terjadi pada bahasa pemrograman yang mendukung overloading metode atau template. Overloading metode adalah kemampuan untuk mendefinisikan beberapa metode dengan nama yang sama, tetapi dengan parameter yang berbeda. Sedangkan, template adalah mekanisme untuk menghasilkan kode secara otomatis dengan menerapkan algoritme yang sama pada tipe data yang berbeda.

2.4.3 Polimorphism Dinamis

Polimorfisme dinamis (*dynamic polymorphism*) adalah jenis polimorfisme dimana metode yang akan dipanggil ditentukan pada saat *runtime* (saat program berjalan), bukan saat kompilasi. Konsep ini juga dikenal dengan nama *late binding* atau *run-time binding*. Polimorfisme dinamis dapat dicapai dalam bahasa pemrograman yang mendukung konsep pewarisan atau interface. Contoh pada bahasa Python, kita dapat menggunakan konsep pewarisan atau *interface* untuk membuat polimorfisme dinamis.

2.5 Interface

Program Berorientasi Objek (OOP) yang berikutnya yaitu interface. Pada subbab ini berisi penjelasan mengenai Pengertian Interface, bagaimana cara mendeklarasikan interface dan implementasi interface, dan pewarisan pada interface.

2.5.1 Pengertian Interface

Interface dalam pemrograman berorientasi objek adalah kontrak atau abstraksi yang mendefinisikan serangkaian metode yang harus diimplementasikan oleh kelas-kelas yang menggunakannya. *Interface* menyediakan pola atau *blueprint* yang menggambarkan perilaku yang diharapkan dari objek tanpa memperhatikan detail implementasi dari metode tersebut.

Interface memungkinkan untuk memiliki kelas-kelas yang berbeda secara struktural tetapi dapat dipergunakan secara bersama-sama dalam polimorfisme, di mana objek dari kelas-kelas yang berbeda dapat dianggap sebagai objek dengan tipe interface yang sama.

2.5.2 Cara Mendeklarasikan Interface

Dalam mendeklarasikan interface pada program ditentukan dengan beberapa langkah-langkah diantaranya:

1. Buatlah sebuah file program baru untuk memulai program
2. Gunakan keyword “*interface*” sebelum nama kelas yang ditentukan.
3. Berikan *method-method* yang nantinya akan di deklarasikan oleh kelas kelas deklarasi *interface*.
4. Setelah interface didefinisikan, kelas yang akan mengimplementasikan *interface* tersebut dapat dideklarasikan. Gunakan kata kunci “*implements*” untuk menunjukkan bahwa kelas tersebut mengimplementasikan *interface*.

2.5.3 Implementasi Interface

Terdapat beberapa cara dalam mengimplementasikan interface diantaranya:

1. Buatlah sebuah nama kelas yang akan di interface kan
2. Menggunakan keyword “*implements*” untuk mendeklarasikan kelas baru terhadap kelas yang menjadi interface yang akan diimplementasikan.

3. Implementasikan method yang didefinisikan pada interface tersebut. Method yang didefinisikan pada kelas yang mengimplementasikan interface tersebut.
4. Instansiasi objek dari kelas yang mengimplementasikan interface tersebut. Objek dapat diinstansiasi seperti objek dari kelas biasa.
5. Panggil method yang didefinisikan pada interface pada objek yang telah diinstansiasi. Karena kelas tersebut mengimplementasikan interface, maka method yang didefinisikan pada interface dapat dipanggil pada objek tersebut.

2.5.4 Pewarisan Pada Interface

Pewarisan atau inheritance pada *interface* dapat dilakukan dengan cara meng-extend atau memperluas sebuah *interface* oleh *interface* lainnya. Langkah-langkah untuk melakukan pewarisan pada *interface* diantaranya:

1. Tentukan interface yang akan dijadikan parent atau induk.
2. Buat *interface* baru dan gunakan kata kunci “*extends*” diikuti dengan nama *interface parent* atau induk tersebut.
3. Definisikanlah beberapa *method* atau suatu variable *instance* yang dibutuhkan pada *subinterface* tersebut.
4. Implementasikan *subinterface* pada kelas yang diinginkan. Kelas yang mengimplementasikan subinterface tersebut harus mengimplementasikan semua *method* yang didefinisikan pada *subinterface* dan juga pada class *parent interface*.

2.6 Encapsulation

Program Berorientasi Objek (OOP) yang berikutnya yaitu encapsulation. Pada subbab ini berisi penjelasan tentang pengertian encapsulation, control akses, public, protected dan tanpa acces specifier.

2.6.1 Pengertian Encapsulation

Encapsulation adalah konsep dalam pemrograman berorientasi objek yang menggabungkan data dan perilaku (metode) yang berhubungan dengan data ke dalam satu unit tunggal yang disebut objek. Ini bertujuan untuk menyembunyikan detail implementasi internal dari objek dan hanya memungkinkan akses terkontrol melalui metode yang ditentukan.

Dalam konteks encapsulation, objek berfungsi sebagai kapsul atau wadah yang menyimpan data (*state*) dan perilaku (*behavior*) yang terkait satu sama lain. Data objek biasanya didefinisikan sebagai variabel-variabel pribadi (*private*), yang artinya hanya dapat diakses oleh metode-metode yang ada di dalam objek itu sendiri. Metode-metode ini bertindak sebagai antarmuka publik (*public*) untuk mengakses dan memanipulasi data tersebut.

2.6.2 Control Akses

Kontrol akses dan enkapsulasi adalah dua konsep utama dalam pemrograman berorientasi objek (OOP) yang digunakan untuk mengorganisasi dan mengatur akses dan data perilaku objek.

1. Kontrol akses adalah mekanisme yang digunakan untuk membatasi akses ke anggota objek seperti variabel dan metode. Hal ini dilakukan untuk menjaga keamanan dan integritas data objek serta mencegah akses yang tidak sah atau tidak diinginkan. Dalam banyak bahasa pemrograman, terdapat tiga tingkat kontrol akses yang umum digunakan:
 - a. *Public*. Anggota *public* dapat diakses dari mana saja di dalam program. Ini berarti anggota tersebut dapat diakses baik dari objek itu sendiri maupun dari objek lain yang menggunakan objek tersebut.
 - b. *Protected*. Anggota dilindungi hanya dapat diakses oleh kelas yang mendefinisikannya dan oleh kelas turunannya. Anggota ini tidak dapat diakses dari luar kelas atau objek.

- c. *Private*. Anggota pribadi hanya dapat diakses oleh kelas yang mendefinisikannya. Anggota ini tidak dapat diakses dari kelas turunan atau dari luar kelas.
- 2. Enkapsulasi. Enkapsulasi adalah konsep dalam OOP yang menggabungkan data dan perilaku yang terkait dalam satu objek tunggal dan menyembunyikan rincian implementasinya dari pengguna objek. Ini dilakukan dengan menggunakan konsep kelas dan objek, di mana variable dan metode yang terkait dengan objek dikapsulkan dalam kelas yang mengelola akses ke mereka.

2.6.3 Tanpa Access Specifier

Tanpa access specifier , encapsulation tidak dapat diterapkan dapat diterapkan secara efektif dalam pemrograman berorientasi objek. Access seperti public, private dan protected sangat penting dalam membatasi akses dan penggunaan variable dalam sebuah objek.

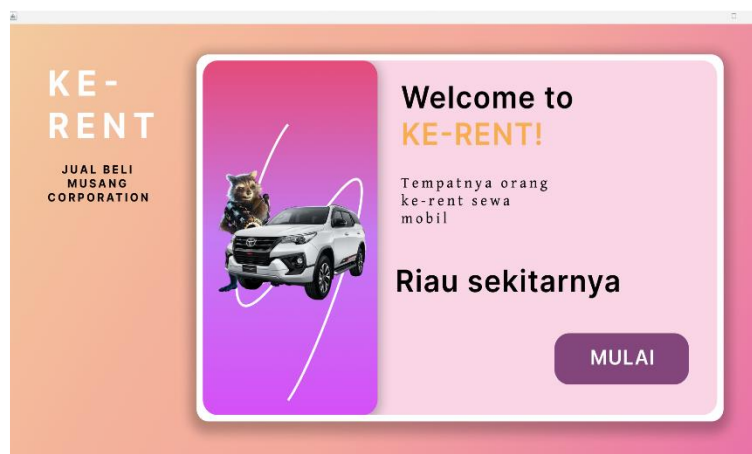
Dalam hal ini, tanpa acces specifier, semua variable dan metode dalam sebuah objek dapat diakses dari mana saja di program, yang dapat menyebabkan masalah dengan integritas data dan penggunaan yang tidak terduga. Selain itu, tanpa acces specifier, kode menjadi lebih sulit untuk dipahami dan diatur, karena tidak jelas mana yang harus diakses dan mana yang tidak.

BAB 3

HASIL DAN PEMBAHASAN

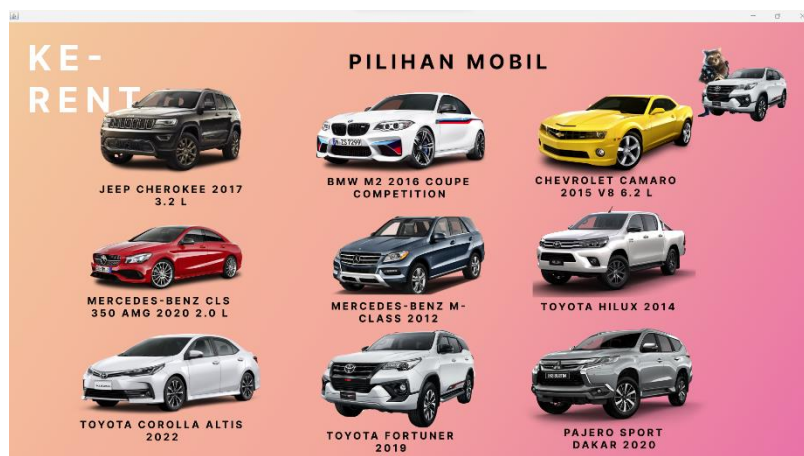
3.1 Tampilan Aplikasi

Pada subbab ini, akan menampilkan tampilan dari project aplikasi rental mobil dengan nama “Ke-Rent”.



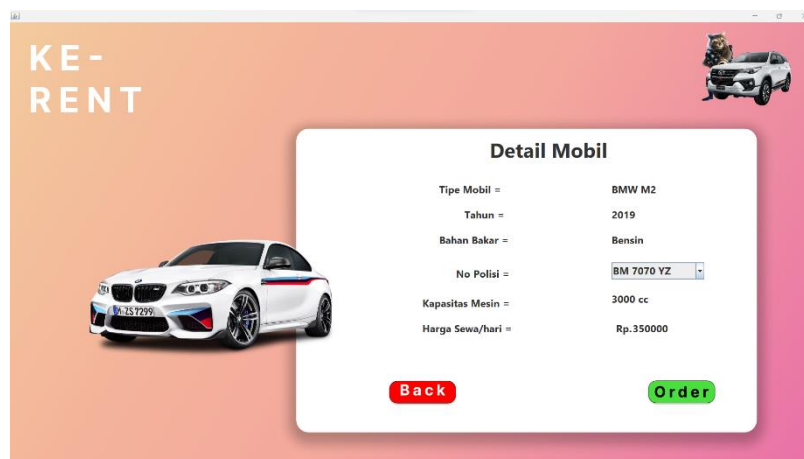
Gambar 1 Tampilan Menu Awal

Pada gambar 1 Tampilan Menu Awal diatas, merupakan tampilan awal halaman dalam aplikasi “Ke-Rent”. Disana terdapat tombol mulai, ketika dijalankan akan diarahkan ke menu pilihan mobil.



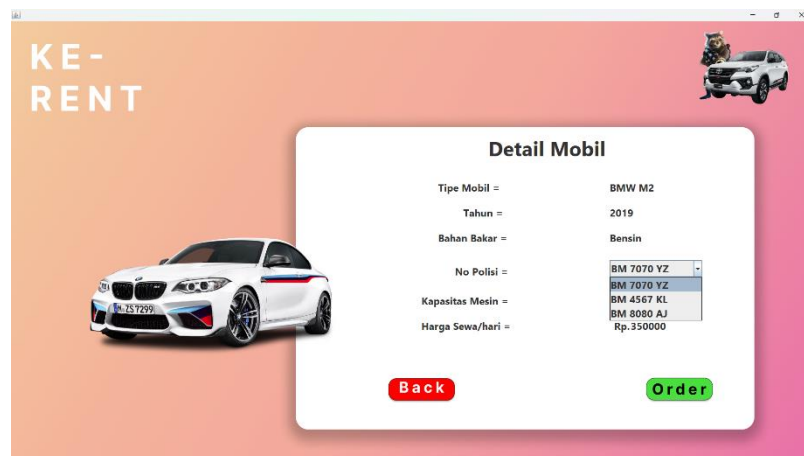
Gambar 2 Tampilan Pemilihan Mobil

Pada gambar 2 Tampilan Pemilihan Mobil, pengguna akan diarahkan ke menu pemilihan mobil. Disana terdapat 9 jenis mobil beserta namanya, ketika pengguna ingin memilih mobil, user hanya perlu menekan gambar mobil yang ingin dipilih, kemudian pengguna akan diarahkan ke halaman yang berisi tentang detail mobil yang dipilih.



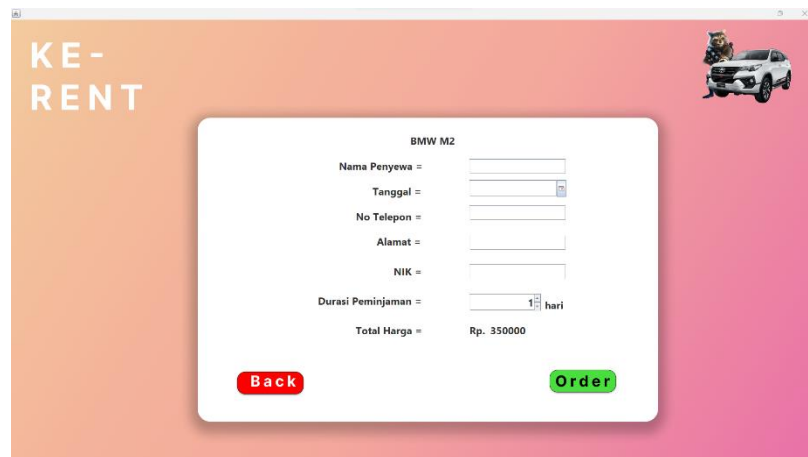
Gambar 3 Detail Mobil

Pada gambar 3 Menu Detail Mobil, terdapat detail dari mobil yang kita pilih pada *frame* sebelumnya. Disana juga terdapat pilihan pada opsi nomor polisi, yang jika ditekan akan muncul beberapa opsi yang bisa dipilih sesuai dengan gambar 4 Pemilihan Nomor Polisi dibawah.



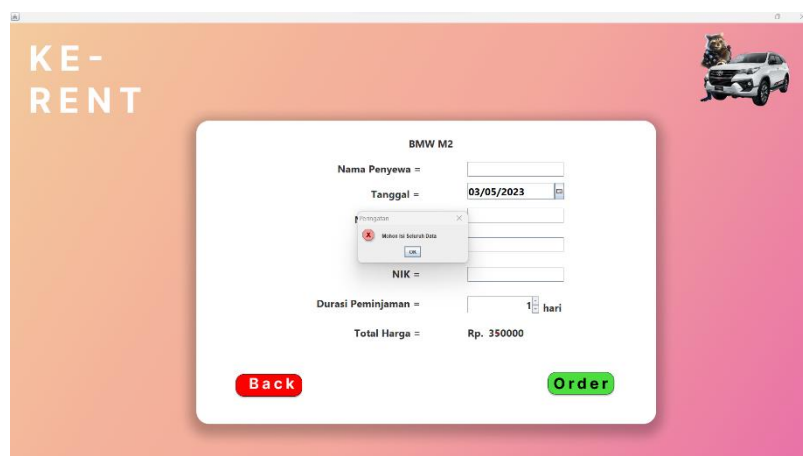
Gambar 4 Pemilihan Nomor Polisi

Ketika user sudah memilih nomor polisi sesuai yang diinginkan, maka dapat menekan tombol “order” untuk lanjut ke halaman pengisian data diri, namun jika pengguna ingin mengganti pilihan mobilnya, pengguna dapat menekan tombol “back” untuk kembali ke halaman seperti pada gambar 2 Tampilan Pemilihan Mobil diatas

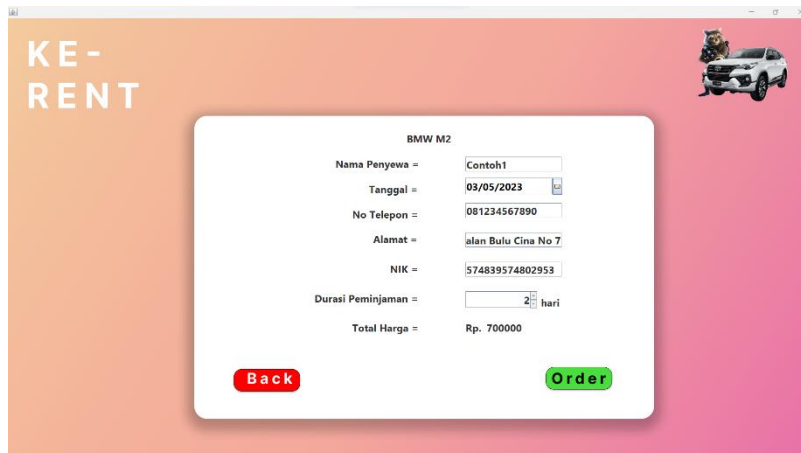
The screenshot shows a web application interface for car rental. The background is a gradient of orange and pink. In the top left, the text 'KE-RENT' is displayed. In the top right, there is a small image of a white car with a person sitting in it. The main content is a white rectangular form titled 'BMW M2'. Inside the form, there are several input fields: 'Nama Penyewa', 'Tanggal' (with a calendar icon), 'No Telepon', 'Alamat', 'NIK', 'Durasi Peminjaman' (set to '1' with a dropdown for 'hari'), and 'Total Harga' (displaying 'Rp. 350000'). At the bottom of the form, there are two buttons: a red 'Back' button and a green 'Order' button.

Gambar 5 Pengisian Data Diri

Pada gambar 5 Pengisian Data Diri diatas terdapat halaman untuk mengisi data diri pengguna. Pengguna dapat mengisi data sesuai yang diminta, pengguna juga dapat memilih berapa hari ingin melakukan sewa mobil. Jika penngguna menekan tombol “order” tetapi belum melengkapi data yang diminta, maka akan muncul notifikasi peringatan seperti gambar 6 Peringatan Isi Data dibawah.

This screenshot is similar to the previous one, but it includes a warning dialog box. The dialog box is titled 'Peringatan' and contains the text 'Mohon Isi Seluruh Data' with an 'OK' button. The form behind it shows that the 'Tanggal' field is now filled with '03/05/2023'. The 'Back' and 'Order' buttons are still visible at the bottom of the form.

Gambar 6 Peringatan Isi Data



KE-RENT

BMW M2

Nama Penyewa = Contoh1

Tanggal = 03/05/2023

No Telepon = 081234567890

Alamat = Jalan Bulu Cina No 7

NIK = 574839574802953

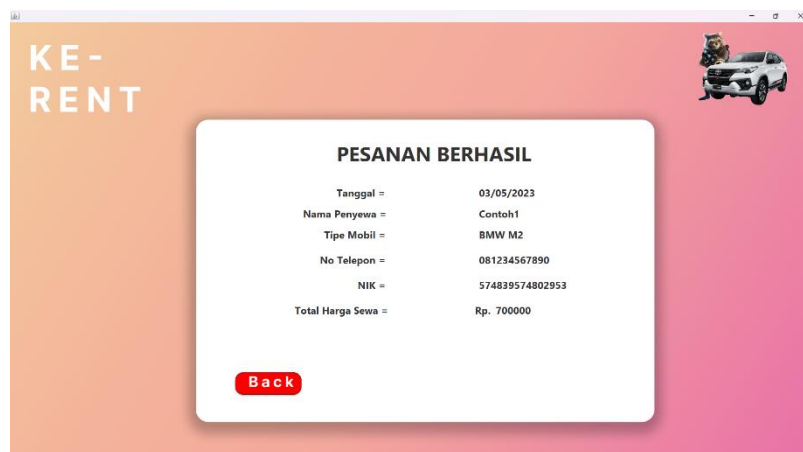
Durasi Peminjaman = 2 hari

Total Harga = Rp. 700000

Back **Order**

Gambar 7 Contoh Pengisian Data

Pada gambar 7 Contoh Pengisian Data diatas, disana terdapat nama penyewa, tanggal yang dapat di tombol sebelah kanan, lalu terdapat nomor telepon dan nik yang hanya bisa di input dengan angka. Disana juga dapat durasi peminjaman yang dapat disesuaikan sesuai keinginan pengguna. Setelah semua diisi, maka pada total harga akan menampilkan nominal yang harus dibayar, jika sudah sesuai pengguna dapat menekan tombol “order” untuk menuju ke halaman konfirmasi pemesanan. Namun jika pengguna ingin memilih mobil kembali, pengguna dapat menekan tombol “back” yang akan menuju ke menu pemilihan mobil sesuai gambar 2 Tampilan Pemilihan Mobil.



KE-RENT

PESANAN BERHASIL

Tanggal = 03/05/2023

Nama Penyewa = Contoh1

Tipe Mobil = BMW M2

No Telepon = 081234567890

NIK = 574839574802953

Total Harga Sewa = Rp. 700000

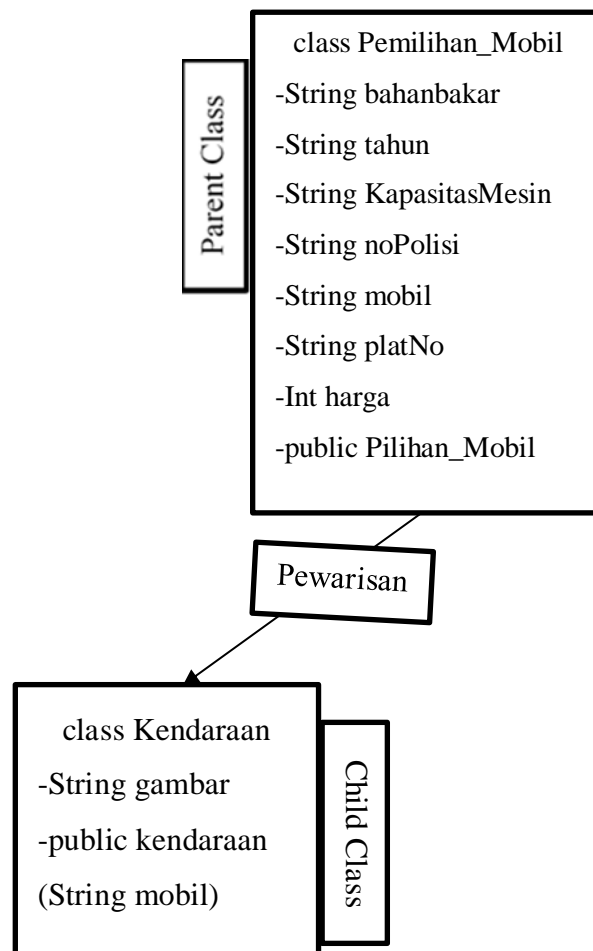
Back

Gambar 8 Pemesanan Berhasil

Pada gambar 8 Pemesanan Berhasil, disana tertera data yang sudah diisi pengguna pada frame sebelumnya dan pemesanan berhasil dilakukan. Jika pengguna ingin memilih mobil kembali, pengguna dapat menekan tombol “back” dan akan kembali ke tampilan pengisian data diri sesuai dengan gambar 5 Pengisian Data Diri.

3.2 Pembahasan Kode Program

Pada subbab ini, penulis akan membahas mengenai kode program (*Coding*) dari project aplikasi rental mobil. Berikut adalah *flowchart* dalam penerapan *inheritance*.

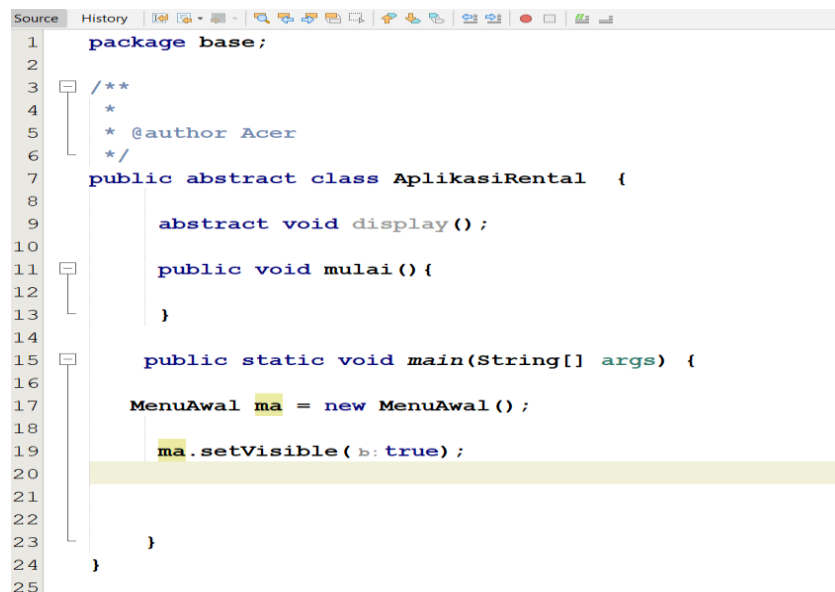


Gambar 9 Flowchart Inheritance

Pada gambar 9 *Flowchart Inheritance*, terdapat dua *class*, yaitu *class* `Pemilihan_Mobil` sebagai *parent class* dan *class* `Kendaraan` sebagai *child class*.

Pada class Pemilihan_Mobil terdapat tipe data *String* dengan atribut bahanbakar, tahun, KapasitasMesin, noPolisi, mobil, platNo dan tipe data Interger dengan atribut harga, juga terdapat satu konstruktor. Pada *child class* terdapat tipe data *String* tambahan dengan atribut gambar dan juga satu *method non void* dengan nama kendaraan berparameter String dengan atribut mobil.

Project program aplikasi rental mobil dimulai dari *abstract class* AplikasiRental, yang didalamnya terdapat *method abstract* dan *method void non abstract*, kita memanggil class MenuAwal dengan mengubah nama objeknya menjadi “ma”. Lalu penulis memasukkan perintah “*ma.setVisible(true)*”, jadi ketika di jalankan, akan langsung membuka tampilan *frame* dari class MenuAwal Sesuai dengan gambar 10 Membuka Aplikasi dibawah dan akan menampilkan frame sesuai dengan gambar 1 Tampilan Menu Awal.



```

1  package base;
2
3  /**
4   *
5   * @author Acer
6   */
7  public abstract class AplikasiRental {
8
9      abstract void display();
10
11      public void mulai() {
12
13      }
14
15      public static void main(String[] args) {
16
17          MenuAwal ma = new MenuAwal();
18          ma.setVisible(true);
19
20      }
21
22
23  }
24
25

```

Gambar 10 Membuka Aplikasi

Lalu, di class MenuAwal yang sudah ter-extends otomatis ke javax.swing.JFrame karena penulis membuat class JFrame. Di dalam class MenuAwal terdapat satu konstruktor dan juga terdapat satu buah label dan satu buah button sesuai dengan gambar 11 kode menu awal dibawah ini.

```

package base;

import javax.swing.JFrame;

/**
 *
 * @author Acer
 */
public class MenuAwal extends javax.swing.JFrame {

    /**
     * Creates new form MenuAwal
     */
    public MenuAwal() {
        initComponents();
        this.setExtendedState( state: JFrame.MAXIMIZED_BOTH);
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    Generated Code

```

Gambar 11 Kode Menu Awal 1

```

private void btn_mulaiActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Pilihan_Mobil pm = new Pilihan_Mobil();
    pm.setVisible(b: true);
    this.dispose();
}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    Look and feel setting code (optional)

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new MenuAwal().setVisible(b: true);
        }
    });
}

// Variables declaration - do not modify
private javax.swing.JButton btn_mulai;
private javax.swing.JLabel jLabel1;
// End of variables declaration

```

Gambar 12 Kode Menu Awal 2

Pada button, penulis mengisi kode yang bertujuan untuk pindah ke frame berikutnya, yaitu class Pemilihan_Mobil dan akan menampilkan frame gambar 2 Tampilan Pemilihan Mobil.

```

package base;

import javax.swing.JFrame;

public class Pilihan_Mobil extends javax.swing.JFrame {

    private String bahanBakar, tahun, kapasitasMesin, noPolisi;
    private String mobil;
    private int harga;
    private String[] platNo;

    /**
     * Creates new form Pilihan_Mobil
     */
    public Pilihan_Mobil() {
        initComponents();
        this.setExtendedState( state:JFrame.MAXIMIZED_BOTH);
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.

```

Gambar 13 Pilihan Mobil 1

```

@SuppressWarnings("unchecked")
Generated Code

private void btn_JeepCherokeeActionPerformed(java.awt.event.ActionEvent evt) {
    mobil = "Jeep Cherokee";
    DetailMobil detailMobil = new DetailMobil(mobil);

    Kendaraan kendaraan = new Kendaraan(mobil);
    kendaraan.setItemsPlatNo(mobil);

    DataClassMobil dataClassMobil = new DataClassMobil();
    platNo = dataClassMobil.getPlatNo();

    detailMobil.setComboboxItem(platNo);
    detailMobil.setVisible(b:true);
    this.dispose();
}

```

Gambar 14 Pilihan Mobil 2

Pada gambar 14 Pilihan Mobil 2, di dalam *button*, penulis mengisi kode program dengan memanggil atribut mobil lalu menginput “ mobil = “nama mobil” ” dan dibawahnya diikuti dengan memanggil *class* DetailMobil dengan nama objek detailmobil, memanggil *class* Kendaraan dengan nama objek kendaraan dan memanggil *class* DataClassMobil dengan nama objek dataclassmobil yang didalamnya kita panggil atribut platNo. Hal yang sama juga dilakukan ke 8 mobil lainnya dan ketika pengguna menekan gambar mobil, maka pengguna akan diarahkan ke *frame* detail mobil sesuai dengan gambar 3 Detail Mobil.

```

package base;

import interfaces.AppInterface;

public class Kendaraan extends Pilihan_Mobil implements AppInterface {

    private String mobil, bahanBakar, tahun, kapasitasMesin, noPolisi, gambar;
    private int harga;
    private String[] platNo;

    public Kendaraan(String mobil) {
        this.mobil = mobil;
        displaySpec();
    }
}

```

Gambar 15 Kendaraan 1

Pada class Kendaraan terdapat penerapan inheritance dan interface, yaitu class Kendaraan meng – extends class Pilihan_Mobil dan mengimplementasikan AppInterface.

```

@Override
public void setDetailMobil(String mobil, String platNo) {
    if ("Jeep Cherokee".equals( anObject:mobil)) {
        gambar = "cherokeee.png";
        switch (platNo) {
            case "BM 7162 CR":
                bahanBakar = "Bensin";
                tahun = "2019";
                kapasitasMesin = "3000 cc";
                noPolisi = "BM 7162 CR";
                harga = 300000;
                break;
            case "BK 1871 VM":
                bahanBakar = "Bensin";
                tahun = "2014";
                kapasitasMesin = "2500 cc";
                noPolisi = "BK 1871 VM";
                harga = 300000;
                break;
            case "BM 3333 X":
                bahanBakar = "Diesel";
                tahun = "2022";
                kapasitasMesin = "3400 cc";
                noPolisi = "BM 3333 ";
                harga = 350000;
                break;
            default:

```

Gambar 16 Kendaraan 2

Pada Gambar 16 Kendaraan 2, terdapat *method Override* yang mengubah isi *method void* setDetailMobil dengan parameter String mobil dan plat no. Di dalam method void tersebut terdapat penerapan *If-Else* dan juga penerapan *Switch Case* dengan menggunakan atribut gambar dan platNo. Jadi didalam *If-Else* tersebut berisi tentang informasi detail beserta 3 platNo yang berbeda, hal itu membuat pengguna bisa memilih platNo yang berbeda. Hal ini juga dilakukan ke 8 mobil lainnya.

```

package base;

import javax.swing.ImageIcon;
import javax.swing.JFrame;

public class DetailMobil extends javax.swing.JFrame {

    private static String bahanBakar, tahun, kapasitasMesin, noPolisi, gambar;
    private static String mobil;
    private static int harga;

    /**
     * Creates new form DetailMobil
     */
    public DetailMobil(String mobil) {
        initComponents();
        this.setExtendedState ( state.JFrame.MAXIMIZED_BOTH );

        this.mobil = mobil;
    }

    public DetailMobil() {
        initComponents();
    }
}

```

Gambar 17 Kode Detail Mobil

Pada gambar 17 Kode DetailMobil berisi tipe data *String* dan *Integer* beserta atribut nya dan juga sebuah konstruktor dengan parameter dan tanpa parameter.

```

private void jMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
    String platNoDipilih = (String) jMenuItem.getSelectedItemAt();
    Kendaraan kendaraan = new Kendaraan(mobil);
    kendaraan.setDetailMobil(mobil, platNoDipilih);

    DataClassMobil dataClassMobil = new DataClassMobil();
    bahanBakar = dataClassMobil.getBahanBakar();
    tahun = dataClassMobil.getTahun();
    kapasitasMesin = dataClassMobil.getKapasitasMesin();
    noPolisi = platNoDipilih;
    harga = dataClassMobil.getHarga();
    gambar = dataClassMobil.getGambar();

    setLabelText(mobil, bahanBakar, tahun, kapasitasMesin, noPolisi, harga, gambar);
}

private void btn_backActionPerformed(java.awt.event.ActionEvent evt) {
    Pilihan_Mobil pm = new Pilihan_Mobil();
    pm.setVisible(b: true);
    this.dispose();
}

private void btn_orderActionPerformed(java.awt.event.ActionEvent evt) {
    TemplateMobil tm = new TemplateMobil(mobil);
    tm.setVisible(b: true);
    this.dispose();
}

```

Gambar 18 Kode Detail Mobil 2

Pada *class* DetailMobil juga terdapat *combo box* dan juga *button* yang berfungsi untuk kembali dan ke menu selanjutnya. Pada *combo box* menampilkan detail mobil beserta nomor polisi yang sudah dipilih tadi beserta keterangan mobilnya. Ketika kita menekan *button* order maka akan diarahkan ke *class* TemplateMobil sesuai dengan gambar 5 Pengisian Data Diri.

```

public class TemplateMobil extends javax.swing.JFrame {

    int harga;
    private static String mobil;
    private static String nama, telepon, NIK, tanggal, alamat;
    private static int totalHarga, durasi;

    public TemplateMobil(String mobil) {
        initComponents();
        this.setExtendedState( state: JFrame.MAXIMIZED_BOTH);

        this.mobil = mobil;
        setMobilText();
        jSpinner1.setValue( value:1);
    }

    private void txt_ValueMobil(java.awt.event.ActionEvent evt) {

    }

    @SuppressWarnings("unchecked")
    Generated Code

    private void txt_ValuePenyewaActionPerformed(java.awt.event.ActionEvent evt) {

    }
}

```

Gambar 19 Kode TemplateMobil 1

Pada *class* TemplateMobil terdapat tipe data *String* dan *Integer* beserta atributnya dengan *control access private*. Di dalamnya juga terdapat konstruktor dengan parameter.

```

private void btn_orderActionPerformed(java.awt.event.ActionEvent evt) {

    DataClassPesan dc = new DataClassPesan();

    DataClassPesan dataClassPesan = new DataClassPesan();
    nama = txt_ValuePenyewa.getText();
    telepon = txt_ValueNoTelepon.getText();
    NIK = txt_ValueNIK.getText();
    SimpleDateFormat date = new SimpleDateFormat( sstring: "dd/MM/yyyy");
    tanggal = String.valueOf( obj: date.format( date: txt_tanggal.getDate()));
    alamat = txt_ValueAlamat.getText();

    totalHarga = Integer.parseInt( s: txt_ValueTotalHarga.getText());

    dataClassPesan.setName(nama);
    dataClassPesan.setTelepon(telepon);
    dataClassPesan.setNIK(NIK);
    dataClassPesan.setTanggal(tanggal);
    dataClassPesan.setAlamat(alamat);
    dataClassPesan.setTotalHarga(totalHarga);

    KonfirmasiPembayaran kp = new KonfirmasiPembayaran(mobil);
}

```

Gambar 20 Kode TemplateMobil2

Pada *button* diatas, penulis mengisi kode agar pengguna bisa memilih waktu seperti tanggal, bulan dan tahun. Diikuti dengan mengubah totalharga yang awalnya *integer* menjadi *string* agar dapat ditampilkan. Data yang sudah diinput tadi dipanggil kembali agar bisa tampil pada halaman selanjutnya, yaitu *class* konfirmasipembayaran. Jika pengguna tidak mengisi seluruh data, maka ketika ingin lanjut ke halaman selanjutnya akan muncul notifikasi dari *JOptionPane*.


```

import javax.swing.JFrame;

public class KonfirmasiPembayaran extends javax.swing.JFrame {

    public static String nama, tanggal, mobil, telepon, NIK;
    int totalHarga;

    public KonfirmasiPembayaran(String mobil) {
        initComponents();
        this.setExtendedState( state: JFrame.MAXIMIZED_BOTH);

        DataClassPesan dataClassPesan = new DataClassPesan();
        DetailMobil detailMobil = new DetailMobil(mobil);
        this.mobil = mobil;

        nama = dataClassPesan.getNama();
        tanggal = dataClassPesan.getTanggal();
        telepon = dataClassPesan.getTelepon();
        NIK = dataClassPesan.getNIK();
        totalHarga = dataClassPesan.getTotalHarga();

        setLabelText(nama, tanggal, telepon, NIK, totalHarga, mobil: KonfirmasiPembayaran.mobil);
    }
}

```

Gambar 21 Kode Konfirmasi

Pada gambar 21 konfirmasi, akan tampil *frame* seperti gambar 8 Pesanan Berhasil. Di dalam class KonfirmasiPembayaran terdapat tipe data String dan Integer beserta atributnya, juga terdapat konstruktor dengan parameter yang didalamnya berisi data yang ingin ditampilkan sebagai *output*.

```

private javax.swing.JLabel jLabel1;
private javax.swing.JLabel txt_DetailMobil;
private javax.swing.JLabel txt_KapasitasMesin;
public javax.swing.JLabel txt_Rp;
private javax.swing.JLabel txt_Tahun;
private javax.swing.JLabel txt_TipeMobil;
public javax.swing.JLabel txt_ValueHarga;
public javax.swing.JLabel txt_ValueKapasitasMesin;
public javax.swing.JLabel txt_ValueNIK;
public static javax.swing.JLabel txt_ValuePenyewa;
private javax.swing.JLabel txt_ValueTanggal;
public javax.swing.JLabel txt_ValueTelepon;
public javax.swing.JLabel txt_ValueTipeMobil;
public javax.swing.JLabel txt_ValueTotalHarga;
private javax.swing.JLabel txt_bahanBakar;
private javax.swing.JLabel txt_harga;
// End of variables declaration

void setLabelText(String nama, String tanggal, String telepon, String NIK, int totalHarga, String mobil) {
    txt_ValuePenyewa.setText( text: nama);
    txt_ValueTanggal.setText( text: tanggal);
    txt_ValueTelepon.setText( text: telepon);
    txt_ValueNIK.setText( text: NIK);
    txt_ValueTipeMobil.setText( text: mobil);
    txt_ValueTotalHarga.setText( text: Integer.toString( totalHarga));
}

```

Gambar 22 Kode Konfirmasi 2

BAB 4

PENUTUP

4.1 Kesimpulan

Laporan project aplikasi rental mobil dengan menggunakan penerapan OOP dalam pembuatannya. Kesimpulan laporan ini adalah sebagai berikut:

1. OOP memungkinkan pengembang untuk memecah kompleksitas proyek menjadi objek-objek yang lebih kecil dan terorganisir. Dalam konteks aplikasi rental mobil, objek-objek seperti mobil, transaksi diwakili sebagai kelas-kelas dalam bahasa pemrograman
2. Penggunaan OOP memungkinkan pengembang untuk menerapkan konsep-konsep seperti enkapsulasi, pewarisan dan polimorfisme. Enkapsulasi memungkinkan pengembang untuk menyembunyikan detail implementasi objek dan memberikan antarmuka yang jelas untuk berinteraksi dengan objek tersebut.
3. Pewarisan memungkinkan pengembang untuk membuat hierarki kelas, di mana kelas-kelas turunan dapat mewarisi sifat-sifat dan perilaku dari kelas induk. Dalam aplikasi rental mobil, contohnya adalah kelas Pilihan_Mobil yang dapat menjadi kelas induk untuk kelas Kendaraan, yang memiliki karakteristik khusus masing-masing.
4. Polimorfisme memungkinkan pengembang untuk menggunakan objek dengan tipe yang sama secara umum, tanpa harus mengetahui tipe objek yang sebenarnya. Dalam konteks aplikasi rental mobil, ini dapat bermanfaat ketika memproses transaksi atau mengelola stok mobil, di mana objek-objek dengan tipe yang berbeda dapat diperlakukan secara seragam.
5. Penerapan OOP pada proyek aplikasi rental mobil membantu meningkatkan modularitas, fleksibilitas, dan pemeliharaan kode. Pembagian proyek menjadi kelas-kelas memudahkan pengembang untuk memahami dan mengubah bagian-bagian aplikasi secara terisolasi tanpa mengganggu bagian lain.

Dalam kesimpulannya, makalah ini menunjukkan bahwa penerapan OOP dalam pengembangan aplikasi rental mobil memberikan manfaat signifikan dalam hal struktur, pemeliharaan, dan fleksibilitas. Dengan menggunakan paradigma OOP, pengembang dapat membangun aplikasi yang lebih efisien, mudah dipelihara, dan dapat berkembang secara lebih baik dalam jangka Panjang

4. 2 Saran

Adapun saran yang dapat diberikan untuk project program aplikasi rental mobil dengan menerapkan OOP adalah sebagai berikut:

1. Sertakan tinjauan yang komprehensif tentang konsep-konsep dasar OOP, seperti enkapsulasi, pewarisan, dan polimorfisme. Berikan definisi yang jelas dan contoh relevan untuk membantu pembaca memahami prinsip-prinsip OOP yang diterapkan dalam proyek aplikasi rental mobil.
2. Bagian ini seharusnya memberikan contoh implementasi konkret dari konsep-konsep OOP dalam bahasa pemrograman yang digunakan. Berikan contoh kode untuk kelas-kelas yang relevan, termasuk contoh penggunaan enkapsulasi, pewarisan, dan polimorfisme. Jelaskan juga alasan di balik penggunaan pola desain atau prinsip desain tertentu yang digunakan dalam implementasi.
3. Diskusikan manfaat yang dihasilkan dari penerapan OOP dalam proyek aplikasi rental mobil, seperti modularitas yang lebih baik, kemudahan pemeliharaan, dan fleksibilitas. Namun, juga sertakan pengakuan terhadap beberapa kelemahan atau tantangan yang mungkin timbul dalam penerapan OOP dalam proyek ini.

DAFTAR PUSTAKA

Oracle Corporation. (2021). Java SE Documentation. Diakses dari:
<https://docs.oracle.com/en/java/javase/index.html> diakses pada 22 Mei 2023

Tutorialspoint. (2021). Java Object-Oriented Programming. Diakses dari:
https://www.tutorialspoint.com/java/java_object_oriented.htm diakses pada 22 Mei 2023

Booch, G. (2007). Object-Oriented Analysis and Design with Applications (3rd Edition). Addison-Wesley Professional diakses dari Open.AI pada 22 Mei 2023

Oracle Corporation. (2021). The Java™ Tutorials - Abstract Methods and Classes. Diakses dari: <https://docs.oracle.com/javase/tutorial/java/IandI/abstract.html> pada 22 Mei 2023

Oracle Java Documentation - Polymorphism:
<https://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html> diakses pada 22 Mei 2023

Oracle Java Documentation - Interfaces:
<https://docs.oracle.com/javase/tutorial/java/concepts/interface.html> diakses pada 22 Mei 2023