Jurnal MAHANDIA
Jurnal Ilmiah Multi Disiplin Ilmu
ISSN: 25448-8899 (Print)
https://mahandia.com/jurnal/index.php/jm/index

PENERAPAN POLIMORFISME UNTUK MENINGKATKAN FLEKSIBILITAS DAN REUSABILITAS DALAM PENGEMBANGAN PERANGKAT LUNAK BERORIENTASI OBJEK

Yeni Dayani[1], Sitti Ardila Ilham[2], Nayla[3*], Fatima. B[4]
[1,2,3,4]Program Studi Teknik Informatika, Fakultas Teknik, Universitas Dayanu Ikhsanuddin
[1,2,3,4]timkolaborasi22@gmail.com
*Penulis Koresponden

diajukan: 25 Juni 2024, diterima: 9 Juli 2024.

Abstrak

Fleksibilitas dan reusabilitas dalam pemrograman berorientasi objek merupakan aspek penting yang perlu ditingkatkan untuk memperbaiki efisiensi dan kualitas pengembangan perangkat lunak. Penerapan polimorfisme menjadi solusi yang tepat untuk meningkatkan fleksibilitas dalam pemograman berorientasi objek. Pada metode, Polimorfisme diterapkan dengan membuat hierarki kelas yang memungkinkan kelas turunan untuk mengimplementasikan metode virtual/abstract tanpa mengubah kode di kelas dasar. Hal ini memungkinkan objek dari kelas yang berbeda untuk digunakan dengan cara yang sama. Hasil dan pembahasan menunjukkan setelah penerapan polimorfisme, fleksibilitas dan reusabilitas dalam pengembangan perangkat lunak mengalami peningkatan. Fleksibilitas terwujud melalui kemampuan objek untuk memiliki perilaku yang berbeda tanpa merubah kelas asli, sementara reusabilitas tercapai dengan penggunaan ulang kode tanpa perlu mengubah logika dasar aplikasi. Kesimpulannya yaitu, penerapan polimorfisme dalam PBO memberikan manfaat nyata dalam meningkatkan fleksibilitas dan reusabilitas perangkat lunak. Dengan polimorfisme, pengembang dapat menciptakan aplikasi yang lebih modular, mudah diperluas, dan efisien dalam pengembangan.
Kata Kunci : Polimorfisme,reusabilitas,fleksibilitas

Abstract

Flexibility and reusability in object-oriented programming are important aspects that need to be improved to improve the efficiency and quality of software development. The application of polymorphism is the right solution to increase flexibility in object-oriented programming. In methods, Polymorphism is implemented by creating a class hierarchy that allows derived classes to implement virtual/abstract methods without changing the code in the base class. This allows objects of different classes to be used in the same way. The results and discussion show that after implementing polymorphism, flexibility and reusability in software development has increased. Flexibility is realized through the ability of objects to have different behavior without changing the original class, while reusability is achieved by reusing code without the need to change the basic logic of the application. The conclusion is that the application of polymorphism in PBO provides real benefits in increasing software flexibility and reusability. With polymorphism, developers can create applications that are more modular, easily extensible, and efficient in development.
Keywords: Polymorphism, reusability, flexibility

1. PENDAHULUAN

Pengembangan perangkat lunak telah mengalami perkembangan pesat selama beberapa dekade terakhir. Salah satu paradigma pengembangan yang menjadi sangat populer adalah perangkat lunak berorientasi objek. Paradigma ini telah mengubah cara kita merancang, mengembangkan dan memelihara perangkat lunak. Perangkat lunak berorientasi objek adalah pendekatan yang didasarkan pada konsep objek. Objek adalah entitas yang memiliki atribut dan perilaku tertentu. Dalam paradigma ini, perangkat lunak direpresentasikan sebagai kumpulan objek yang saling berinteraksi untuk mencapai tujuan tertentu. Pendekatan ini memungkinkan pemodelan yang lebih intuitif dan fleksibel, serta mempermudah pengelolaan kompleksitas perangkat lunak. Beberapa keuntungan utama dari pengembangan perangkat lunak berorientasi objek adalah reusabilitas dan fleksibilitas. Dengan memanfaatkan konsep polimorfisme, pengembang dapat membuat kelas-kelas yang dapat digunakan kembali dalam berbagai konteks. Hal ini dapat mengurangi redundansi kode dan meningkatkan efisiensi pengembangan. Pengembang juga dapat memanfaatkan konsep enkapsulasi untuk melindungi data dan metode dari akses langsung yang tidak diinginkan, meningkatkan keamanan dan integritas kode.

Polimorfisme terbagi menjadi dua suku kata, yaitu *Poly* yang berarti banyak dan *morfisme* yang berarti bentuk. Dalam ilmu sains, polimorfisme adalah sebuah prinsip dalam biologi dimana organisme atau spesies memiliki banyak bentuk serta tahapan (*stages*). Dalam pengembangan perangkat lunak berorientasi objek, polimorfisme merupakan sebuah konsep dimana *class* memiliki banyak "bentuk" *method* yang berbeda, meskipun namanya sama. Maksud dari "bentuk" adalah isinya berbeda, namun namanya sama. Polymorfisme juga dapat diartikan sebagai teknik *progamming* yang mengarahkan untuk memprogram secara general daripada secara spesifik. Contohnya saat memiliki tiga *class* yang berbeda yaitu, "kelinci", "kucing", dan "Sapi". Dimana ketiga *class* tersebut merupakan turunan dari *class* "Hewan".

Konsep Polimorfisme terkait dengan pewarisan atau *inheritance*. Pewarisan memungkinkan kita untuk membuat kelas baru yang mewarisi sifat dan perilaku dari kelas yang ada. Dalam hal ini, kelas baru dapat memiliki metode atau perilaku yang sama dengan kelas yang di warisi, atau dapat mengganti perilaku yang sudah ada dengan perilaku yang baru. Konsep polimorfisme memungkinkan kita untuk membuat program yang lebih modular dan fleksibel. Objek dapat memiliki perilaku yang berbeda-beda tergantung pada jenis atau tipe yang sedang digunakan. Kita dapat mengubah perilaku objek tanpa perlu merubah kelas asli dari objek tersebut. Ada dua jenis jenis polimorfisme, antara lain:

1) Polimorfisme waktu kompilasi

Polimorfisme waktu kompilasi terjadi saat metode yang dipanggil ditentukan pada waktu kompilasi. Jenis objek yang dipanggil harus diketahui pada waktu kompilasi, contohnya adalah *method overloading*. *Method overloading* adalah salah satu bentuk polimorfisme yang terjadi pada tahap kompilasi, dimana kita dapat mendefinisikan beberapa *method* dengan nama yang sama namun memiliki perbedaan parameter yang diterima, baik dalam jumlah maupun jenisnya. Teknik ini memungkinkan kita untuk membuat *method* yang fleksibel dan lebih mudah di baca, karena penggunaannya dapat disesuaikan dengan kebutuhan spesifik tanpa harus membuat nama *method* yang berbeda-beda untuk setiap variasi parameter. Dengan *overloading*, kita dapat meningkatkan keterbacaan dan manajemen kode, serta memberikan cara yang lebih elegan untuk menangani berbagai tipe input yang berbeda dalam sebuah kelas.

2) Polimorfisme waktu berjalan

Polimorfisme waktu berjalan terjadi saat metode yang dipanggil ditentukan pada waktu berjalan. Dalam hal ini, jenis objek yang di panggil dapat ditentukan saat program berjalan. Contohnya adalah method *overriding*. *Method overriding* memberikan kemampuan bagi kelas anak untuk menyusun kembali atau menggantikan implementasi *method* yang telah di definisikan dikelas induknya dengan versi yang lebih sesuai untuk konteks atau kebutuhan spesifik kelas

tersebut. Dengan melakukan *overriding* kelas anak dapat memanfaatkan kembali struktur *method* dari kelas induk sambil memberikan perilaku yang unik atau lebih spesifik. Hal ini memungkinkan pewarisan fitur yang lebih dinamis dan memberikan fleksibilitas yang lebih besar dalam memodifikasi perlaku objek tanpa harus mengubah kode yang sudah ada dikelas induk. *Overriding* merupakan salah satu pilar dari prinsip *Object Oriented Progamming* yang memungkinkan polimorfisme dinamis, sehingga *method* yang di panggil ditentukan pada waktu berjalan, tergantung pada tipe objek yang memanggil metode tersebut.

Dalam dunia pengembangan website yang semakin kompleks dan dinamis, penggunaan konsep pemograman berorientasi objek telah menjadi landasan yang esensial. Pemograman berorientasi objek membawa keefektifan, kemudahan pemeliharaan, dan skalabilitas yang ternilai harganya untuk proyek-proyek webiste. Salah satu manfaat utama pemograman berorientasi objek adalah kemampuannya untuk memecah kode menjadi modul atau objek yang independen yang disebut modularitas dan pengelolaan kode yang lebih baik. Dalam pengembangan website, setiap komponen seperti formulir, tampilan, database, dan lainnya dapat diwakili sebagai objek yang terpisah. Ini memungkinkan tim pengembang untuk bekerja secara terpisah pada komponen-komponen ini, mempercepat proses pengembangan dan memungkinkan pemeliharaan yang lebih mudah dimasa depan. Selain itu, dalam pemograman berorientasi objek, objek-objek dapat digunakan ulang diberbagai bagian proyek yang biasa disebut *Reusability* atau penggunaan ulang. Ini mengurangi jumlah kode yang ditulis, menghemat waktu dan usaha pengembang. Misalnya, jika telah membuat objek formulir yang memiliki metode untuk memvalidasi input, kita dapat menggunakannya diberbagai halaman website tanpa perlu menulis ulang kode validasi tersebut. Serta pemograman berorientasi objek memungkin untuk membuat abstraksi yang tinggi untuk mengelola kerumitan dan mendefinisikan pola umum yang disebut fleksibilitas. Ini memungkinkan kualitas kode karena mengikuti prinsip-prinsip yang terbukti dalam desain perangkat lunak, seperti DRY (*Don't Repeat Yourself*) dan SOLID (Prinsip responsibilitas terpisah, terbuka-tertutup, substitusi Liskov, Segregasi antarmuka, ketergantungan inversi).

Dalam pengembangan perangkat lunak berorientasi objek, seringkali terjadi tantangan dalam meningkatkan fleksibilitas dan reusabilitas. Reusabilitas berarti pernyataan yang memungkinkan pengembang untuk menulis fungsionalitas tertentu sekali dan menggunakannya berkali-kali sedangkan fleksibilitas kemampuan untuk menggunakan kembali kode yang sudah ada. Untuk itu, penelitian ini akan memberikan pemahaman tentang bagaimana menerapkan prinsip polimorfisme untuk meningkatkan fleksibilitas dan reusabilitas dalam pengembangan perangkat lunak berorientasi objek.

Menurut Erliyan Redi Susanto, M.Kom. (2024),polimorfisme adalah salah satu kelebihan utama pengembangan perangkat lunak berorientasi objek yang memungkinkan objek untuk memiliki beberapa bentuk dan perilaku yang berbeda. Polimorfisme juga dapat memungkinkan pengembang untuk membuat kelas-kelas dasar yang dapat digunakan kembali dalam berbagai konteks,meningkatkan reusabilitas dan fleksibilitas dalam pengembangan perangkat lunak berorientasi objek. Penelitian berjudul Pengukuran *Object Oriented Concepts* pada Aplikasi, penelitian ini menggunakan metrik Chidamber dan Kemerer untuk mengukur kualitas perangkat lunak berorientasi objek. Polimorfisme dianggap sebagai salah satu faktor yang mempengaruhi kualitas perangkat lunak, dengan WMC (*Weighted Method Count*) sebagai indikator yang memprediksikan waktu dan usaha yang diperlukan untuk mengembangkan aplikasi. Menurut Adi Kusjani, S.T., M.Eng. (2024),polimorfisme adalah salah satu kelebihan utama paradigma pemograman berorientasi aspek (AOP) yang memungkinkan pengembang untuk memisahkan aspek-aspek *cross-cutting* dari kode fungsional utama, meningkatkan modularitas, reusabilitas, dan *maintainability* kode.

Menurut Nusabot (2023),polimorfisme memungkinkan objek untuk memiliki banyak bentuk dan perilaku yang berbeda, meningkatkan fleksibilitas dan reusabilitas dalam pengembangan perangkat lunak berorientasi objek. Penelitian berjudul pengembangan sistem

informasi berbasis polimorfisme, pada penelitian ini pemograman berorientasi objek memungkinkan pengembangan sistem yang lebih fleksibel dan reusabilitas. Polimorfisme memungkinkan objek untuk digunakan di berbagai konteks, meningkatkan reusabilitas dan fleksibilitas dalam pengembangan perangkat lunak. Penelitian lain yang berjudul Analisis dan perancangan sistem E-Business berbasis CRM, ini memungkinkan pengembangan sistem yang lebih fleksibel dan reusabilitas. Polimorfisme memungkinkan pengembangan sistem yang lebih mudah diubah dan dipindahkan antar sistem.

Penelitian lain yang berjudul pengembangan sistem informasi berbasis polimorfisme, ini menunjukkan pengembangan sistem yang fleksibel dan reusabilitas. Dalam hal ini, objek dapat memiliki perilaku yang berbeda-beda tergantung pada jenis atau tipe yang sedang digunakan. Dengan cara ini, kita dapat mengubah perilau objek tanpa perlu merubah kelas asli dari objek tersebut. Penelitian lain yang berjudul Enkapsulasi, pewarisan, dan polimorfisme: Pilar utama pmrograman berorientasi objek. Ini menunjukkan penggunaan kode yang sama untuk menangani berbagai jenis objek. Manfaat polimorfisme meliputi code reusability, readability, dan maintainability. Contoh penerapan polimorfisme meliputi penggunaan metode yang berbeda-beda tergantung pada jenis objek.

Polimorfisme dalam pengembangan perangkat lunak berorientasi objek dapat mencapai reusabilitas dan fleksibilitas dengan cara berikut:

1. Dalam Reusabilitas,kode yang ditulis dalam pendekatan berorientasi objek dapat digunakan kembali dalam proyek lain. Ini menghemat waktu dan upaya dalam pengembangan aplikasi baru,serta memastikan konsistensi dan keandalan kode.
2. Dan pada Fleksibilitas,perangkat lunak berorientasi objek,objeknya dapat diperluas dan dimodifikasi dengan mudah tanpa menganggu bagian lain dari aplikasi. Hal ini memungkinkan pengembang untuk dengan cepat menyesuaikan aplikasi dengan perubahan kebutuhan atau persyaratan baru.

Berdasarkan uraian di atas,maka penulis akan menunjukkan melalui studi kasus atau implementasi nyata mengenai bagaimana penerapan polimorfisme dapat secara signifikan meningkatkan reusabilitas dan fleksibilitas dalam proyek pengembangan perangkat lunak tertentu serta memberikan bukti yang mendukung teori ini.

2. METODE

Penelitian ini dilakukan dengan metode studi kasus. Studi kasus adalah metode penelitian suatu kasus yang setiap prosesnya dilakukan secara rinci dan detail. Kasus yang diambil dalam penelitian ini adalah aplikasi manajemen produk kecantikan untuk pengembangan perangkat lunak menggunakan aplikasi delphi versi 11.3. Dalam aplikasi manajemen produk kecantikan, polimorfisme diterapkan melalui desain kelas yang memungkinkan penggunaan metode yang sama untuk berbagai jenis produk kecantikan. Tujuan pengembangan aplikasi ini yaitu meningkatkan reusabilitas dan fleksibilitas melalui penerapan polimorfisme. Polimorfisme diterapkan dengan beberapa cara sebagai berikut:

1) Desain Kelas Dasar (*Abstract Class*)

Membuat sebuah kelas dasar abstrak yang akan menjadi kerangka kerja untuk semua jenis produk kecantikan. Di dalam kelas dasar ini, didefinisikan antarmuka dasar dan metode yang harus diimplementasikan oleh semua subkelas. Seperti metode *Apply* sebagai metode virtual yang harus di *override* oleh sub kelas.

2) Desain Subkelas

Membuat subkelas yang mewarisi kelas dasar *TBeautyProducts* dan didalam setiap subkelas lakukan *overriding* terhadap metode *Apply* sesuai dengan perilaku yang unik dari masing-masing jenis produk kecantikan.

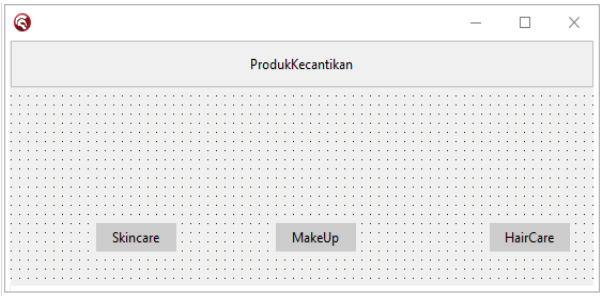
3) Implementasi metode *Apply*

Jurnal MAHANDIA
Jurnal Ilmiah Multi Disiplin Ilmu
ISSN: 25448-8899 (Print)
https://mahandia.com/jurnal/index.php/jm/index

Pastikan metode Apply yang di definisikan dalam kelas dasar *TBeautyProducts* merupakan metode virtual yang diimplementasikan ulang (*override*) oleh setiap subkelas. Setiap subkelas harus mengimplementasikan metode *Apply* sesuai dengan logika yang sesuai dengan jenis produk kecantikan yang bersangkutan.

4) Penggunaan Polimorfisme

Dalam aplikasi, gunakan polimorfisme dengan memanggil metode *Apply* pada objek produk kecantikan tanpa perlu mengetahui jenis spesifik dari objek tersebut. Polimorfisme akan memastikan bahwa metode yang dipanggil adalah versi yang sesuai dari subkelas objek yang digunakan. Dengan menerapkan polimorfisme seperti di atas, aplikasi manajemen produk kecantikan dapat mencapai reusabilitas dan fleksibilitas yang tinggi, karena memungkinkan penambahan jenis produk baru tanpa menganggu logika dasar aplikasi dan memastikan bahwa setiap jenis produk dapat memiliki perilaku yang unik sesuai dengan kebutuhan. Komponen yang digunakan dalam membuat aplikasi manajemen produk kecantikan adalah tiga button dan satu panel.



Gambar 1: Antarmuka Pengguna

Pengembangan aplikasi ini menggunakan kelas dasar dan subkelas serta satu *MainForm*. Kelas dasar *'TBeautyProducts'* akan mendefinisikan antarmuka dasar dan metode *'Apply'* untuk semua jenis produk, dan subkelas *'TSkincare'*, *'TMakeUp'*, *'THairCare'* akan meng-override metode *'Apply'* untuk memberikan implementasi masing-masing jenis produk kecantikan . Metode *'Apply'* di definisikan sebagai metode virtual yang harus di implementasikan oleh subkelas.

3. HASIL DAN PEMBAHASAN

Berikut adalah hasil dan pembahasan dari studi kasus penerapan polimorfisme dalam pengembangan aplikasi manajemen produk kecantikan yang bertujuan meningkatkan reusabilitas dan fleksibilitas di Delphi.

3.1. Desain dan Implementasi kelas

Desain dan implementasi kelas dasar, subkelass dan mainform dalam Delphi antara lain sebagai berikut:

BeautyProducts | HairCare | MakeUp | Skin
TSkincare

```
unit BeautyProducts;

interface

uses Vcl.Dialogs;

type
  TBeautyProduct = class
  public
    procedure Apply; virtual; abstract;
end;

  TSkincare = class(TBeautyProduct)
  public
   constructor Create;
   procedure Apply; override;
end;

  TMakeUp = class(TBeautyProduct)
  public
   constructor Create;
   procedure Apply; override;
end;

  THairCare = class(TBeautyProduct)
  public
   constructor Create;
   procedure Apply; override;
end;
```

BeautyProducts | HairCare | MakeUp | Skincare
TBeautyProduct

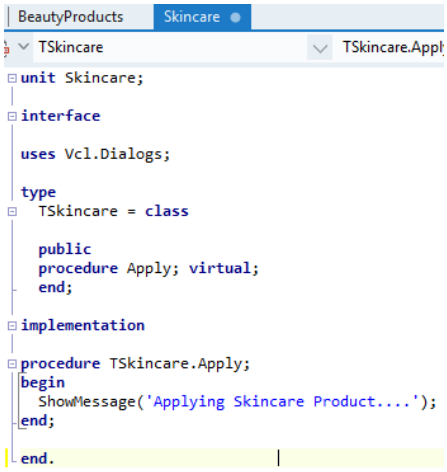
```
implementation

constructor TSkincare.Create;
begin
  inherited;
end;

constructor TMakeUp.Create;
begin
  inherited;
end;

constructor THairCare.Create;
begin
  inherited;
end;

procedure TSkincare.Apply;
begin
  ShowMessage('Applying Skincare Product.....');
end;

procedure TMakeUp.Apply;
begin
  ShowMessage('Applying MakeUp Product.....');
end;

procedure THairCare.Apply;
begin
  ShowMessage('Applying HairCare Product.....');
end;

end.
```

Gambar 2: Program kelas Dasar 1

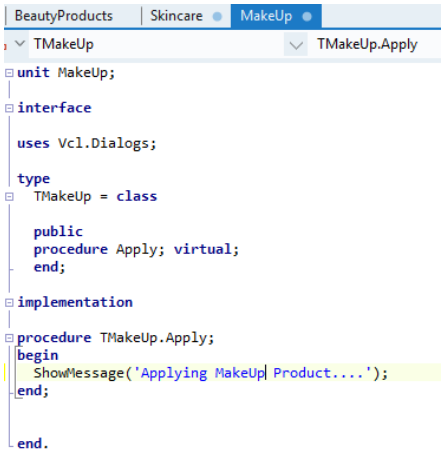
Gambar 3: Program kelas Dasar 2

Berikut penjelasan tiap baris program Kelas Dasar 1 dan 2:

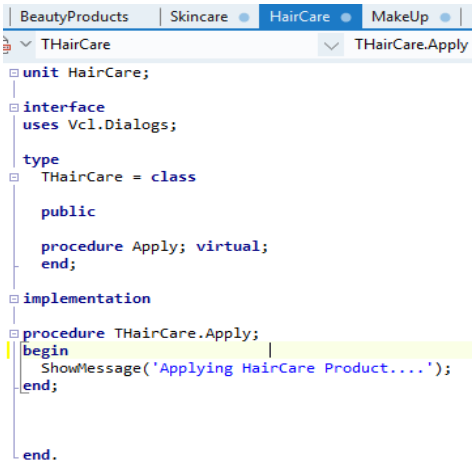
1. Baris pernyataan *uses Vcl.Dialogs;* adalah bagian yang menyediakan dialog standar VCL seperti pesan dialog.
2. Baris kata kunci type digunakan untuk mendefiniskan tipe class *'TBeautyProduct'*.
3. Baris pernyataan *'Public'* sampai *'end;'* menunjukkan bahwa metode dan variabel procedure *'Apply: virtual: abstract;'* masing-masing kelas dapat diakses diluar kelas. Deklarasi procedure *'Apply'* yang bersifat virtual dan *'abstract'*, berarti metode ini diimplementasikan oleh subclass dan tidak memiliki implementasi di kelas dasar.Baris Pernyataan dibawahnya, masing-masing mendefinisikan kelas *'TSkincare'*, *'TMakeUp'*, dan *'THairCare'* yang merupakan subclass dari *'TBeautyProduct'*. Dan mendeklarasikan konstruktor *Create* dan Metode *Apply* yang menimpa *override*.
4. Baris implementasi.Pada bagian ini,yaitu mendefinisikan konstruktor *create* untuk kelas 'TSkincare, TMakeUp, ThairCare' dan memanggil konstruktor dari kelas dasar menggunakan *inherited* untuk memastikan bahwa inisialisasi benar dari kelas dasar.
5. Baris procedure kelas turunan yang mendefinisikan metode *'Apply'* untuk masing-masing kelas turunan. Serta memanggil *'ShowMessage'* untuk menampilkan pesan. Setiap kelas turunan mengimplementasikan metode Apply yang menampilkan pesan spesifik menggunakan *ShowMessage*. Metode ini menimpa metode abstrak *Apply* yang dideklarasikan dikelas dasar *TBeautyProduct*s.



Gambar 4: Program Kelas Turunan Skincare



Gambar 5: Program Kelas Turunan Skincare



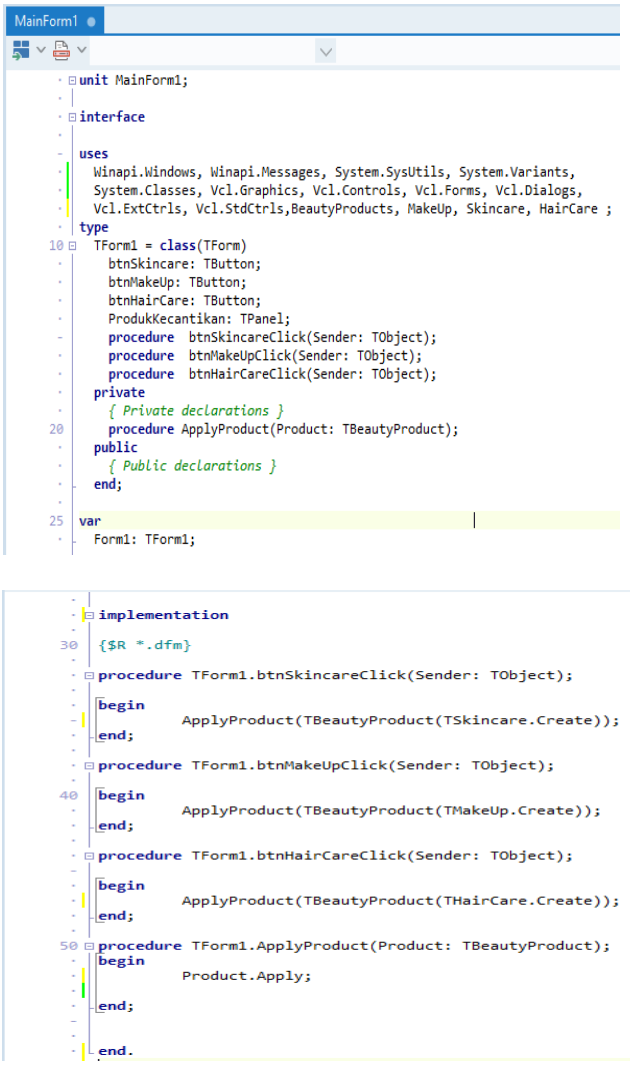
Gambar 6: Program Kelas Turunan Skincare

Berikut penjelasan tiap baris program Kelas Turunan:

1. Baris pernyataan *uses Vcl.Dialogs;* adalah bagian yang menyediakan dialog standar VCL seperti pesan dialog.
2. Baris type (Tkelasturunan) = class, memulai deklarasi tipe.
3. Baris *public* sampai *end;* yaitu mendeklarasikan procedure publik *'Apply'* yang bersifat virtual, yang artinya prosedur ini dapat di *override* oleh kelas turunan ini.
4. Baris implementasi yaitu mengimplementasikan dari prosedur *'Apply'* milik kelas turunan dan menampilkan pesan dengan *'ShowMessage'*.Di akhiri *end;*.

3.2. Interaksi dengan User Interface (MainForm)

MainForm akan menggunakan kelas-kelas produk kecantikan untuk menampilan pesan sesuai metode *'Apply'*. Masing-masing subkelas/kelas turunan akan mewarisi dari kelas dasar *'TBeautyproducts'* dan mengoverride metode *Apply*.



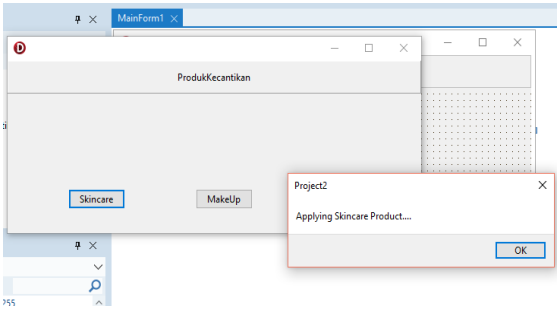
Gambar 7: Program MainForm

Berikut penjelasan tiap baris progam *MainForm* :

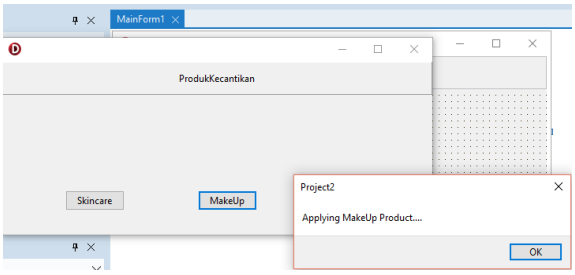
1. Baris *uses*, impor semua kelas dasar dan turunan agar dapat terhubung dan tidak menyebabkan error.
2. Baris *Type*, terdapat semua komponen yang dibutuhkan.
3. Baris *Private*, terdapat procedure *'ApplyProduct(Product: TBeautyProduct);'* ini mendeklarasikan prosedur *ApplyProduct* yang menerima satu parameter *'Product'* dari tipe *'BeautyProduct'*. Prosedure ini hanya dapat di panggil dari dalam kelas *MainForm* saja.
4. Baris Procedure *'TForm1.btnSkincareClick(Sender: Tobject);'* mendeklarasikan prosedure event handler yang dipanggil ketika tombol *'btnSkincare'* di klik. Begitupun juga dengan procedure *'btnMakeUp'* dan *'btnHairCare'*.
5. Deklarasi *'ApplyProduct(TBeautyProduct(TSkincare.Create));'* akan memanggil prosedur *'ApplyProduct'* dengan parameter berupa objek *'TSkincare'*. Dalam arti lain prosedur ini akan membuat instansi dari '*TSkincare'* dan mengonversinya menjadi *'TBeautyProducts'* lalu menerapkannya melalui prosedure *'ApplyProduct'*. Begitupun juga dengan kelas turunan '*MakeUp*' dan '*HairCare*'.
6. Deklarasi *'TForm1.ApplyProduct(Product: TBeautyProduct);* yaitu mendeklarasikan prosedur *'ApplyProduct'* yang menerima paramater Product bertipe *'TBeautyProducts'*. Baris *Begin*, terdapat *'Product.Apply'* yang memanggil metode *'Apply'* pada objek *'Product'*.

3.3. Hasil Pembuatan aplikasi

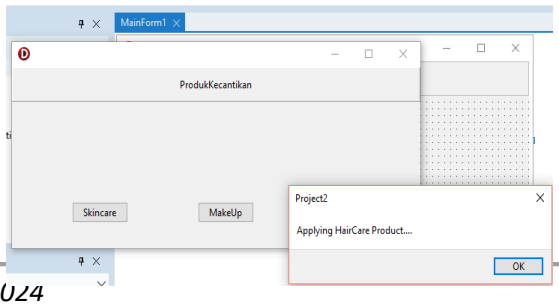
Setelah semua program kelas dan mainform lengkap, saatnya mencoba aplikasi dengan *running*.



Gambar 8: Hasil Running Button Skincare



Gambar 9: Hasil Running Button MakeUp



Gambar 10: Hasil Running Button HairCare

3.4. Penerapan Polimorfisme

Penerapan Polimorfsime dapat dilihat dari adanya kelas dasar dan turunan, metode *virtual/abstract* dan metode *Apply* yang diimplementasikan dimasing-masing kelas tanpa mengubah kode di kelas dasar atau di *MainForm*. Ini memungkinkan *MainForm* untuk menggunakan produk kecantikan dengan cara yang sama meskipun dari kelas yang berbeda. Setelah penerapan polimorfisme dalam pengembangan aplikasi manajemen produk kecantikan menggunakan Delphi, fleksibilitas dan reusabilitas mengalami peningkatan. Berikut adalah penjelasan mengenai peningkatan fleksibilitas dan reusabilitas setelah penerapan polimorfisme:

1. Peningkatan Reusabilitas
   Dengan adanya polimorfisme, kelas dasar abstrak *TBeautyProducts* yang mendefinisikan metode *Apply* memungkinkan penggunaan ulang kode tanpa perlu melakukan perubahan pada kode yang sudah ada. Setiap subkelas dapat mengimplementasikan metode *Apply* sesuai dengan kebutuhan produk kecantikan tanpa mengubah logika dasar aplikasi. Hal ini memungkinkan penggunaan kembali kode untuk berbagai jenis produk kecantikan, sehingga efisiensi dalam pengembangan aplikasi meningkat.
2. Peningkatan Fleksibilitas
   Dengan polimorfisme, aplikasi dapat dengan mudah diperluas dengan menambahkan kelas produk kecantikan baru tanpa menganggu logika dasar aplikasi. Setiap kelas produk kecantikan baru yang merupakan turunan dari kelas dasar *TBeautyProducts* dapat bekerja dengan baik tanpa memerlukan modifikasi tambahan pada prosedur *ApplyProduct*. Polimorfisme memungkinkan objek untukdiperluas dan dimodifikasi dengan mudah tanpa memengaruhi bagian lain dari aplikasi, sehingga pengembang dapat dengan cepat menyesuaikan aplikasi dengan perubahan kebutuhan atau persyaratan baru.

Polimorfisme dapat meningkatkan fleksibilitas dan reusabilitas dalam pengembangan perangkat lunak berorientasi objek. Berikut adalah beberapa poin yang dapat dijelaskan lebih mendalam, yaitu:

1. Reusabilitas melalui Polimorfisme
   Polimorfisme memungkinkan pengembang untuk membuat kelas-kelas yang dapat digunakan kembali dalam berbagai konteks tanpa menulis ulang kode yang sama. Dengan adanya polimorfisme, objek dapat memiliki perilaku yang berbeda-beda tergantung jenis atau tipe yang sedang digunakan. Contohnya adalah ketika sebuah objek dapat digunakan ulang dalam berbagai bagian proyek tanpa perlu mengubah logika dasar aplikasi. Hal ini mengurangi redundansi kode, meningkatkan efisiensi pengembangan dan memastikan konsistensi dalam penggunaan fungsionalitas yang sama.
2. Fleksibilitas melalui polimorfisme
   Polimorfisme memungkinkan objek untuk memiliki beberapa bentuk dan perilaku yang berbeda, sehingga memungkinkan pengembang untuk dengan mudah mengubah perilaku objek tanpa merubah kelas asli dari objek tersebut. Dalam pengembangan perangkat lunak berorientasi objek, polimorfisme memungkinkan pengembang untuk membuat program yang lebih modular. Objek dapat memiliki perilaku yang berbeda-beda tergantung pada jenis atau tipe yang sedang digunakan, sehingga memungkinkan penyesuaian cepat dengan perubahan kebutuhan.
3. Penerapan Polimorfisme dalam studi kasus
   Dalam studi kasus aplikasi manajemen produk kecantikan, penerapan polimorfisme memungkinkan peningkatan reusabilitas dengan adanya kelas abstrak yang mendefinisikan antarmuka umum yang harus diimplementasikan semua kelas turunan. Fleksibilitas juga

ditingkatkan dengan kemudahan ekstensi dimana perubahan kelas produk kecantikan dapat dilakukan tanpa mengubah logika dasar aplikasi. Studi kasus atau implementasi nyata dapat memberikan bukti yang mendukung teori bahwa penerapan polimorfisme secara signifikan dapat meningkatkan reusabilitas dan fleksibilitas dalam proyek pengembangan.

4. KESIMPULAN

Penelitian ini telah menghasilkan bahwa aplikasi manajemen produk kecantikan yang menerapkan polimorfisme dapat meningkatkan reusabilitas dan fleksibilitas. Berikut beberapa kesimpulan yang penulis simpulkan:

1) Polimorfisme merupakan konsep penting dalam pengembangan perangkat lunak berorientasi objek yang memungkinkan objek untuk memiliki beberapa bentuk dan perilaku yang berbeda. Penerapan polimorfisme dapat meningkatkan reusabilitas dan fleksibilitas dalam pengembangan perangkat lunak.

2) Dalam aplikasi manajemen produk kecantikan, penerapan polimorfisme memungkinkan peningkatan reusabilitas dengan adanya kelas dasar abstrak yang mendefinisikan antarmuka umum yang harus diimplementasikan semua kelas. Fleksibilitas juga ditingkatkan dengan kemudahan ekstensi, dimana penambahan kelas produk kecantikan dapat dilakukan tanpa mengubah logika dasar dari aplikasi.

3) Pengembangan perangkat lunak berorientasi objek memberikan keuntungan utama berupa reusabilitas dan fleksibilitas. Dengan memanfaatkan konsep polimorfisme, pengembang dapat membuat kelas-kelas yang dapat digunakan kembali dalam berbagai konteks, mengurangi redundansi kode dan meningkatkan efisiensi pengembangan.

4) Polimorfisme juga dapat membantu dalam memisahkan aspek-aspek *cross-cutting* dari kode fungsional utama, meningkatkan modularitas, reusabilitas, dan *maintainbility* kode dalam paradigma pemograman berorientasi aspek.

5) Penggunaan konsep pemrograman berorientasi objek, seperti polimorfisme, enkapsulasi dan pewarisan dapat membantu meningkatkan keamanan dan integritas kode.

6) Polimorfisme memungkinkan pengembang untuk membuat kelas-kelas yang dapat digunakan kembali dalam berbagai konteks, sehingga mengurangi redundansi kode dan meningkatkan efisiensi pengembangan.

7) Dalam pengembangan website, pemrograman berorientasi objek memungkinkan pemecahan kode menjadi modul atau objek independen, meningkatkan modularitas, pengelolaan kode yang lebih baik, dan reusabilitas kode.

8) Penerapan polimorfisme dalam aplikasi manajemen produk kecantikan berhasil meningkatkan reusabilitas dan fleksibilitas, serta memungkinkan penyesuaian cepat dengan perubahan kebutuhan atau persyaratan baru.

Dengan demikian, konsep polimorfisme dan pemrograman berorientasi objek secara umum memberikan manfaat yang signifikan dalam pengembangan perangkat lunak, termasuk peningkatan reusabilitas, fleksibilitas, keamanan modularitas, dan efisiensi pengembangan.

REFERENSI

"priledu studio," 7 Juni 2024. [Online]. Available: https://www.priludestudio.com/2024/06/menerapkan-konsep-polimorfisme-pada-bahasa-java-html. [Accessed 22 Juni 2024].

"parsinta," 08 Agustus 2023. [Online]. Available: https://parsinta.com/articles/pemrograman-berorientasi-objek-di-php-hzcxd. [Accessed 22 Juni 2024].

"Universitas Islam AN NUR LAMPUNG," 13 Februari 2024. [Online]. Available: https://an-nur.ac.id/enkapsulasi-pewarisan-dan-polimorfisme-pilar-utama-pemrograman-berorientasi-objek/. [Accessed 22 Juni 2024].

Y. Adhawiyah, "Universitas Islam Negeri Syarif Hidayatullah," Oktober 2016. [Online]. Available:

https://repository.uinjkt.ac.id/dspace/bitstream/12345678/66086/1/YUNITA%20ADHAWIYAH%20-%20FST.pdf. [Accessed 2023 Juni 2024].

L. RIAWENNY, "Universitas Islam Negeri Syarif Hidayatullah," Juni 2011. [Online]. Available: https://repository.uinjkt.ac.id/dspace/bitstream/123456789/1554/1/lesly%20riawenny-fst.pdf. [Accessed 23 Juni 2024].

N. N. Hikmah, "NUSA BOT," NUSA BOT, 6 Juli 2023. [Online]. Available: https://nusabot.id/blog/perangkat-lunak-berorientasi-objek-paradigma-pengembangan-masa-kini. [Accessed 9 Juni 2024].

R. Julianto, "dicoding," Dicoding Indonesia, 17 February 2021. [Online]. Available: https://www.dicoding.com/blog/pengertian-polimorfisme-dalam-pemograman-java. [Accessed 9 Juni 2024].

A. C. Putri, "domainesia," DomaiNesia, 21 Mei 2024. [Online]. Available: https://www.domainesia.com/berita/method-adalah/. [Accessed 9 Juni 2024].

M. Erliyan Redi Susanto, "teknokrat," Universitas Teknokrat Indonesia, [Online]. Available: https://ftik.teknokrat.ac.id/pengembangan-perangkat-lunak-menggunakan-metode-terstruktur-vs-berorientasi-objek-memilih-pendekatan-yang-tepat-2/. [Accessed 9 Juni 2024].

H. HANIFAH, "Digilib Unila," 2023. [Online]. Available: https://digilib.unila.ac.id/74091/3/3.%20SKRIPSI%20TANPA%20PEMBAHASAN.pdf. [Accessed 9 Juni 2024].

S. M. Adi Kusjani, "utdi," Universitas Teknologi Digital Indonesia, 1 Maret 2024. [Online]. Available: https://www.utdi.ac.id/terbitan/112/mengenal=paradigma-pemrograman-berorientasi-aspek. [Accessed 9 Juni 2024].

S. N, "pemburu kode," 13 04 2023. [Online]. Available: https://pemburukode.com/polimorfisme-java-konsep-implementasi/. [Accessed 23 Juni 2024].