# PRAKTIKUM

# ALGORITMA DAN STRUKTUR DATA

# TUGAS BESAR


**Disusun Oleh :**

1. Irma Octavia Chaniago (DS-01-01 - 1206210002)
2. Inge Faradila Efaranti (DS-01-01 - 1206210017)
3. Ananda Taqhsya Dwiyana (DS-01-01 - 1206210019)
4. Dida Ardzis Hanana (DS-01-01 - 1206210021)
5. Febrian Farda Hasdikiyah (DS-01-01 - 1206210023)

**PROGRAM STUDI SAINS DATA**

**FAKULTAS TEKNOLOGI INFORMASI DAN BISNIS**

**INSTITUT TEKNOLOGI TELKOM SURABAYA**

**2022**

# Kata Pengantar

Puji syukur kita panjatkan kehadirat Allah Swt. yang telah memberikan rahmat dan hidayah-Nya sehingga kami dapat menyelesaikan tugas besar yang berjudul “Laporan Algoritma struktur Data” ini tepat pada waktunya.

Adapun tujuan dari penulisan dari laporan ini adalah untuk memenuhi tugas besar pada mata kuliah algoritma struktur data. Terlebih dahulu, saya mengucapkan terima kasih kepada Ibu Regita Putri Permata, S.Stat., M.Stat. selaku Dosen Algoritma struktur data yang telah memberi bimbingan dan kepercayaan. Sehingga, laporan ini dapat kami susun dengan baik.

Semoga Laporan ini dapat bermanfaat terutama untuk kami. Walaupun laporan ini masih jauh dari kata sempurna, mengingat kurangnya pengetahuan dan pengalaman kami.

Surabaya, Juli 2022
Penulis

# DAFTAR ISI

# BAB I
# PENDAHULUAN

## 1.1 Latar Belakang

Pemrograman berorientasi objek (Object Oriented Programming atau OOP) merupakan paradigma pemrograman yang berorientasikan kepada objek. Objek adalah struktur data yang terdiri dari bidang data dan metode bersama dengan interaksi mereka untuk merancang aplikasi dan program komputer. Semua data dan fungsi di dalam paradigma ini di bungkus dalam kelas-kela atau objek-objek. Bandingkan dengan logika pemrograman terstruktur, setiap objek dapat menerima pesan, memproses data, dan mengirim pesan ke objek lainnya. Pada jaman sekarang, banyak bahasa pemrograman yang mendukung OOP. OOP adalah paradigma pemrograman yang cukup dominan saat ini, karena mampu memberikan solusi kaidah pemrograman modern. Meskipun demikian, bukan berarti bahwa pemrograman prosedural sudah tidak layak lagi. OOP diciptakan karena dirasakan masih adanya keterbatasan pada bahasa pemrograman tradisional.

Konsep dari OOP sendiri adalah semua pemecahan masalah dibagi ke dalam objek. Dalam OOP data dan fungsi-fungsi yang akan mengoperasikannya digabungkan menjadi satu kesatuan yang dapat disebut sebagai objek. Proses perancangan atau desain dalam suatu pemrograman merupakan proses yang tidak terpisah dari proses yang mendahului, yaitu analisis dan proses yang mengikutinya. Pembahasan mengenai orientasi obejk tidak akan terlepas dari konsep objek seperti inheritance atau penurunan, encapsulation atau pembungkus, dan polymorphism atau kebanyakan rupaan. Konsep-konsep ini merupakan fundamental dalam orientasi objek yang perlu sekali dipahami serta digunakan dengan baik, dan menghindari penggunaan yang tidak tepat. Model data berorientasi objek dikatakan dapat memberi fleksibilitas yang lebih, kemudahan mengubah program, dan digunakan luas dalam teknik piranti lunak skala besar. Lebih jauh lagi, pendukung OOP mengklaim bahwa OOP lebih mudah dipelajari bagi pemula dibanding dengan pendekatan sebelumnya, dan pendekatan OOP lebih mudah dikembangkan dan dirawat.

## 1.2 Rumusan Masalah

Dari latar belakang tersebut, kami dapat menuliskan rumusan masalah sebagai berikut, program untuk mengelompokkan hewan berdasarkan jenisnya, menggunakan konsep Polymorphism serta menyimpan objek ke dalam database.

## 1.3 Tujuan

Adapun tujuan dari laporan ini adalah Untuk memenuhi Tugas besar Algoritma Struktur Data.

## 1.4 Manfaat Penelitian

Manfaat yang didapatkan adalah memahami sejarah, pengertian, konsep dasar, dan istilah-istilah OOP. Disini memperselesaikan masalah menggunakan konsep Polymorphism.

## 1.5 Batasan Masalah

Berdasarkan rumusan masalah, maka kami membatasi permasalahan agar lebih fokus dan terarah, batasan masalah ini adalah :

1. Penyelesaian kasus yang diberikan dengan menggunakan OOP dan konsep polymorphism
2. Menggunakan bahasa pemrograman python

# BAB II
# DASAR TEORI

## 2.1 Sejarah OOP

Sejarah perkembangan OOP dimulai pada tahun 1966 saat Ole Johan Dhal dan Kristen Nygaard dari Universitas Olslo Norwegia menerbitkan sebuah jurnal kertas kerja dengan judul "SIMULASI An Algol Based Language". Pemrograman Berorientasi Pbjek (OOP) merupakan metode pemrograman dimana pengembang harus mendefinisikan tipe dari struktur data dan juga tipe dari operasi yang dapat di aplikasikan ke struktur data, dengan demikian struktur data menjadi objek yang dapat memiliki data dan fungsi. Beberapa kemampuan utama dari pemrograman OOP antara lain :

1. Pemrograman OOP menekankan pada data dari pada prosedur karena data diperlakukan sebagai elemen yang penting dan tidak boleh mengalir secara bebas dalam program.
2. Data di sembunyikan dari akses program oleh fungsi-fungsi (function) eksternal.
3. Program dapat dibagi-bagi kedalam objek-objek yang lebih kecil.
4. Objek dapat berkomunikasi satu dengan yang lain melalui function.
5. Data baru dan function dapat dengan mudah ditambahkan pada saat di butuhkan.
6. Konsep pemrogramannya mengikuti pendekatan buttom up.

## 2.2 Pengertian OOP

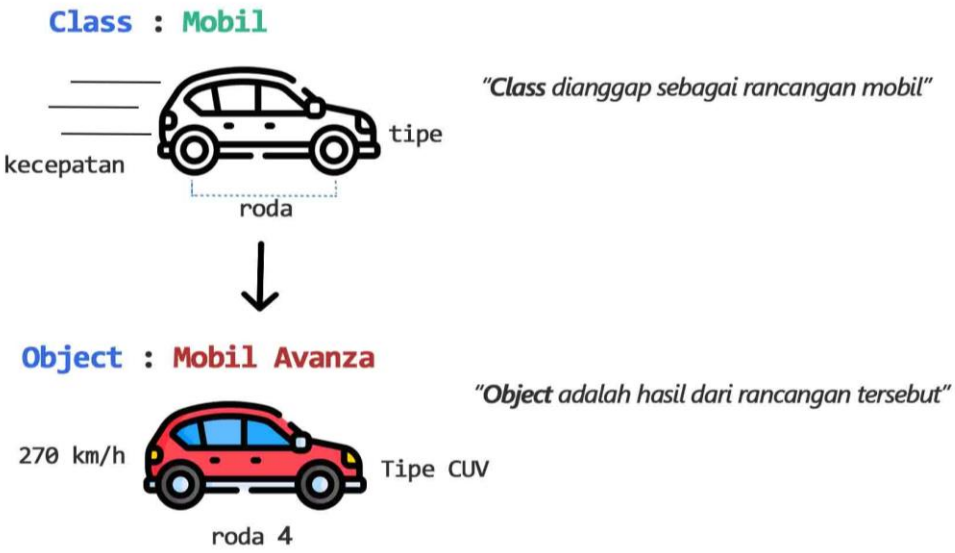
OOP adalah paradigma atau Teknik pemrograman dimana semua hal dalam program dimodelkan seperti objek dalam dunia nyata. Objek di dunia nyata memiliki ciri atau attribute dan juga aksi atau kelakuan (behaviour).

Kita misalkan sebuah mobil. Mobil memiliki ciri punya ban, stang, kursi, pedal gas, rem, dan lain sebagainya. Ada juga ciri warna, atau tahun keluaran berapa. Selain punya ciri, mobil juga punya aksi atau sesuatu yang bisa dilakukan olehnya. Misalnya, ketika pedal diinjak apa yang terjadi. Ketika di rem apa yang terjadi, dan lain sebagainya.

Program juga demikian. Semua unit dalam program bisa dianggap sebagai objek. Objek besar dibangun dari objek – objek yang lebih kecil. Objek yang satu berinteraksi dengan objek yang lain, sehingga semua menjadi sebuah kesatuan yang utuh. Python dari awal dibuat sudah mengadopsi OOP. Selain itu Python juga bisa menggunakan paradigma pemrograman lama yaitu pemrograman terstruktur. Oleh karena itu, Python disebut bersifat hibrid.

Contoh : Kita punya *Class* mobil yang memiliki attribute roda, kecepatan dan tipe. Serta didalam mobil dia juga memiliki method untuk melaju(), klakson() dan

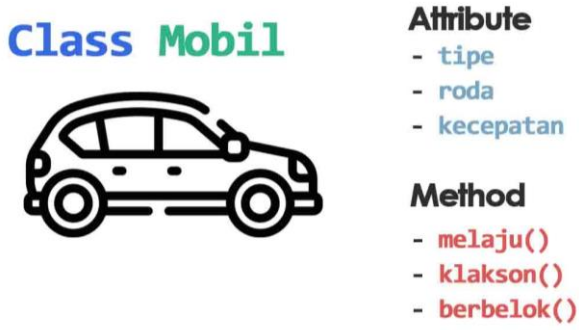
berbelok(). Jika *class* dalam perumpamaannya adalah blueprint dari suatu kategori yang umum, maka object diumpamakan sebagai hal yang lebih spesifik. Jika, *class* mobil tadi bisa membuat banyak objek yang lebih spesifik, misal mobil Avanza atau mobil ferari.




## 2.3 Konsep Dasar

1. **Class**

   *Class* adalah *'cetakan'*, *'template'* atau *'blueprint'* untuk membuat sebuah **object**. *Class* hanya sebagai kerangka dasar dari **object**. Sehingga nanti cetakan *(class)* tersebut dapat membuat berbagai macam **object** untuk kebutuhan yang berbeda-beda.
   *Class* diibaratkan sebagai kategori yang umum, seperti yang dijelaskan diatas. Anggaplah kita membuat *Class* Bernama "mobil", didalam *class* tersebut terdapat karakteristik (*attribute*) dan perilaku (*method*) yang hanya dimiliki oleh mobil.



Dalam bentuk code python :

```
class Mobil:

    def __init__(self, roda, tipe, kecepatan)
        self.tipe = tipe
        self.roda = roda
        self.kecepatan = kecepatan

    def doMelaju(self):
        print("Melaju dengan kecepatan : ", self.kecepatan)

    def doKlakson(self):
        print("klakson")

    def doBelok(self, arah):
        print("Belok arah ", arah)
```

## 2. Attribute

*Attribute* atau *properties* merupakan karakteristik dari sebuah *class*. *Attribute* ini berupa suatu variabel didalam sebuah *class*. Variabel yang didefinisikan sebagai sebuah attribute juga disbut dengan variabel **Global.**

Cara mendeklarasikan sebuah attribute sama dengan cara mendeklarasikan sebuah variabel. Dalam python, variabel yang digunakan sebagai attribute biasanya didefinisikan dalam fungsi constructur (__init__) dan menggunakan keyword self. Variabel attribute pada class disebut juga dengan **instance (turunan) variable**.

```
def __init__(self):
    self.roda = None
    self.tipe = None
    self.kecepatan = 0
```

Pada kode diatas bisa dibilang kita mendeklarasikan attribute dengan nilai default (None dan 0) menggunakan fungsi constructor (__init__) pada class.

Lalu kita bandingkan dengan kode dibawah ini.

```
def __init__(self, roda, tipe, kecepatan):
    self.roda = roda
    self.tipe = tipe
    self.kecepatan = kecepatan
```

Pada kode diatas, nilai pada instance variabel didefiniskan mengikuti parameter yang diberikan pada fungsi constructor (__init__).

Penerapan kedua kode diatas tergantung dengan kondisi pada saat pendeklarasian sebuah object, apakah object ini perlu parameter atau tidak. Jika tidak maka pakai kode yang pertama, jika iya pakai kode yang kedua.

### 3. Method

Method adalah fungsi yang didefinisikan dalam suatu class. Biasanya method memiliki hubungan dalam behaviour atau perilaku kelas tersebut. Dalam kasus class mobil, anbi mendefinisikan 3 method yaitu melaju, klakson dan berbelok.

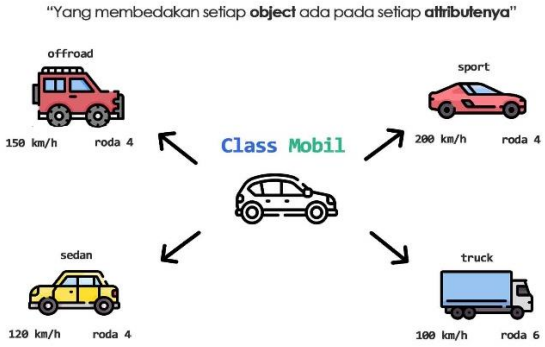
Penulisan method dalam python, hampir sama seperti kita mendefinisikan sebuah fungsi pada umumnya. Cuman, yang membedakan, disetiap mendefinisikan sebuah method parameter keyword self harus selalu ada. Paramater self selalu terdapat pada awal parameter disetiap method. Nah, cara contoh penulisan method dalam python kurang lebih seperti ini.

```
def doMelaju(self):
    print("Melaju dengan kecepatan ", self.kecepatan, "  Km/h")

def doKlakson(self):
    print("klakson")

def doBelok(self, arah):
    print("Belok arah ", arah)
```

### 4. Object

Membungkus data dan fungsi bersama menjadi suatu unit dalam sebuah program computer, object merupakan dasar dari modularitas dan struktur dalam sebuah program computer berorientasi object.

Object adalah instance atau representasi dari sebuah class. Jika class adalah sebuah cetakan, maka object adalah hasil dari cetakan tersebut. Kita dapat membuat berbagai object dengan class mobil yang tadi kita buat.




Cara mendeklarasikan object dari sebuah class pada python adalah dengan memanggil nama class beserta dengan parameter yang diberikan pada fungsi constructor (__init__).

```
mobilFerari = new Mobil("Sport", 4, 200)
mobilJeep = new Mobil("Offroad", 6, 150)
```


Pada kode diatas membuat 2 object pertama adalah object mobilFerari dan mobilOffroad. Kedua object tersebut dibuat dari class yang sama (Mobil) dan memiliki perilaku (behaviour) yang sama tapi dengan karakteristik (attribute) yang berbeda.
Cara mengaksesnya hanya dengan memberikan titik . dalam setiap nama object. Lihat kode dibawah.

```
# akses attribute tipe dari mobilFerari
print(mobilFerari.tipe)
# output : Sport

# akses attribute tipe dari mobilOffroad
print(mobilJeep.tipe)
# output : Offroad
```

Kita bisa menyimpulkan bahwa setiap object memiliki nilai tipe yang berbeda sesuai dengan saat dideklarasikan. Lalu coba kita mengakses method doMelaju() dan kita coba lihat bagaimana output disetiap object.

```
   # memanggil method domelaju di mobilFerari
mobilFerari.doMelaju()
# Output : Melaju dengan kecepatan 200 Km/h

# memanggil method domelaju di mobilOffroad
mobilJeep.doMelaju()
# Output : Melaju dengan kecepatan 150 Km/h
```

Jika kita lihat kode pada method doMelaju() terdapat attribute self.kecepatan. Sehingga setiap object memiliki output yang berbeda dengan perilaku yang sama yaitu “melaju”.

## 5. Data Abstraction

Abstraksi data adalah konsep yang menyembunyikan detail latar belakang dan hanya mewakili informasi yang diperlukan untuk dunia luar. Ini adalah proses penyederhanaan konsep dunia nyata menjadi komponen yang mutlak diperlukan.
Contoh sesuatu yang disebut abstraksi adalah Anda hanya tahu bagian-bagian penting untuk naik sepeda, seperti roda, rem, gear, tanpa menyertakan proses latar belakang atau penjelasan (akselerasi, pengereman, dan sebagainya).

6. **Data Encapsulation**

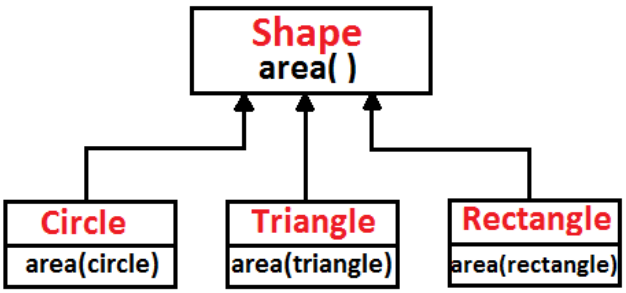
Enkapsulasi data adalah salah satu konsep terpenting dari OOP. Ini adalah teknik yang menggabungkan data members dan fungsi, beroperasi pada data itu dalam satu unit yang dikenal sebagai kelas. Teknik ini pada dasarnya mencegah akses ke data secara langsung. Satu-satunya cara untuk mengakses data disediakan oleh fungsi.

Jika Anda ingin membaca data dalam suatu objek, Anda harus memanggil fungsi anggota dalam objek. Fungsi akan membaca data dan mengembalikan data kepada Anda. Jadi, Anda tidak memiliki akses ke data secara langsung. Karena data disembunyikan, data tersebut diamankan dari perubahan yang tidak disengaja.

7. **Polymorphism**

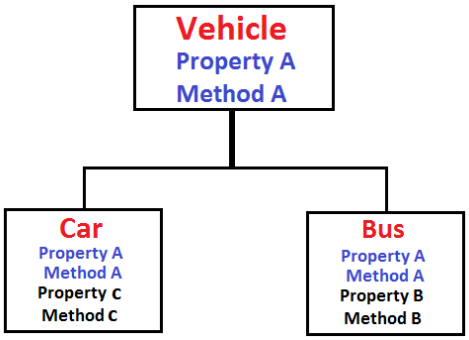
Ini adalah kunci kekuatan OOP. Jika suatu bahasa tidak mendukung polimorfisme maka itu bukan dalam kategori bahasa OOP. Pada dasarnya polimorfisme adalah kemampuan suatu pesan atau data untuk diproses lebih dari satu bentuk. Ini adalah konsep penting dari pemrograman berorientasi objek yang mendukung kemampuan suatu objek kelas untuk berperilaku berbeda dalam menanggapi pesan atau tindakan.

Melalui pengiriman pesan. Tidak bergantung kepada pemanggilan subrutin, bahasa orientasi objek dapat mengirim pesan; metode tertentu yang berhubungan dengan sebuah pengiriman pesan tergantung kepada objek tertentu di mana pesa tersebut dikirim. Contohnya, bila sebuah burung menerima pesan "gerak cepat", dia akan menggerakan sayapnya dan terbang. Bila seekor singa menerima pesan yang sama, dia akan menggerakkan kakinya dan berlari. Keduanya menjawab sebuah pesan yang sama, namun yang sesuai dengan kemampuan hewan tersebut. Ini disebut polimorfisme karena sebuah variabel tungal dalam program dapat memegang berbagai jenis objek yang berbeda selagi program berjalan, dan teks program yang sama dapat memanggil beberapa metode yang berbeda di saat yang berbeda dalam pemanggilan yang sama. Hal ini berlawanan dengan bahasa fungsional yang mencapai polimorfisme melalui penggunaan fungsi kelas-pertama.



**8. Inheritance**

Warisan juga merupakan karakteristik penting dari OOP. Ini pada dasarnya adalah metode yang menyediakan cara yang kemampuan dan properti dari satu kelas untuk datang ke kelas lain. Teknik ini memberikan penggunaan kembali kode untuk programmer. Kita bisa membentuk kelas baru dari kelas yang ada, di mana kelas yang ada berisi beberapa properti atau metode yang juga ada di kelas baru. Di sini kelas baru disebut sebagai kelas turunan. Sedangkan kelas yang ada yaitu kelas dari mana kelas baru diturunkan disebut sebagai kelas dasar.




**9. Overloading**

Overloading juga merupakan konsep kunci lain dari bahasa OOP. Dalam satu baris, kita dapat mendefinisikan beban berlebih karena merupakan kemampuan fungsi tunggal untuk melakukan tugas yang berbeda bergantung pada situasinya. Jadi, konsep overloading entah bagaimana terkait dengan properti polymorphism OOP. Ketika fungsi atau operator yang ada akan dioperasikan pada tipe data baru, itu disebut kelebihan beban (overloaded).

Overloading memungkinkan membuat metode yang berbeda dengan nama yang sama yang mana berbeda satu sama lain dalam jenis fungsi input dan output. Ini dapat digunakan dengan fungsi dan anggota.

**10. Coupling**

Coupling adalah tingkat ketergantungan/hubungan kode program terhadap kode progra lainnya.

Dalam pemrograman, berorientasi obyek tingkat coupling dapat diminimalisasi, sehingga memudah kita dalam memprogram.

**11. Subclass**

Subclass merupakan class yang diwarisi sifat-sifat dari superclass. Ketika class diturunkan dari class yang lain, maka object hasil turunan disebut dengan subclass.

Dalam pemrograman, subclass menyediakan state/ behaviour yang spesifik yang membedakan dengan superclass, sehingga memungkinkan programmer untuk menggunakan ulang source code dari superclass yang telah ada.

**12. Superclass**

Superclass merupakan class yang mewariskan atribut dan methodnya ke subclass. Dapat juga diartikan sebagai class yang menurunkan menjadi class lain (kebalikan dari subclass).

**13. Instance**

Pembentukan dari class menjadi object dikenal dengan istilah Instance. Dapat dijelaskan lebih lanjut bahwa, class tersebut belum dapat digunakan jika hanya sekedar cetak biru dari sebuah object karena belum terbentuk, oleh karena itu class harus dijadikan object agar dapat digunakan maka ini lah penerapannya dalam pemrograman.

**14. Behavior**

Adalah hal-hal yang bisa dilakukan oleh objek dari suatu class. Dalam pemrograman, behavior dapat digunakan untuk mengubah nilai atribut suatu objek, menerima informasi dari objek lain, dan mengirim informasi ke objek lain untuk melakukan suatu tugas (task).

# BAB III
# METODOLOGI PENELITIAN

## 3.1 Sumber Data

Sumber data penelitian ini dilakukan pada tanggal 4 Juli 2022. Penelitian ini menggunakan Object Oriented Programming (OOP) dan konsep polymorphism. Sumber data dari laporan ini menginput dokumen tertulis yang telah kami analisis yang digunakan dalam menyelesaikan permasalah yang diberikan.

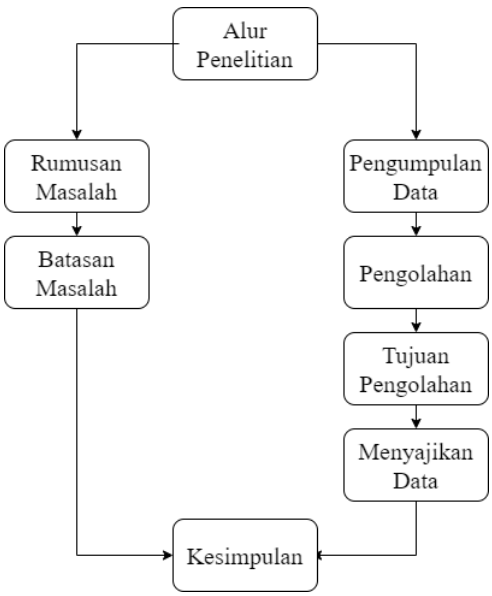
## 3.2 Langkah Analisis

Di Dalam tahap analisis sistem terdapat langkah-langkah dasar yang harus dilakukan oleh analis sistem :

1. Rumusan masalah
2. Tinjauan pustaka
3. Sumber data
4. Alur penelitian
5. Analisis dan pembahasan
6. Kesimpulan

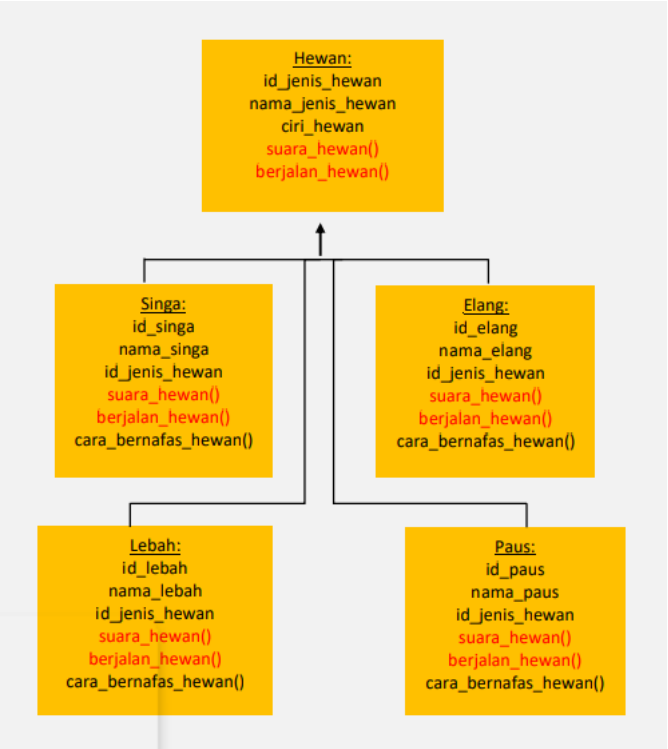
## 3.3 Alur Penelitian

Alur penelitian pada makalah ini memiliki beberapa tahap yaitu :



# BAB IV
# ANALISIS DAN PEMBAHASAN

3. Buatlah program untuk mengelompokkan hewan berdasarkan jenisnya, diantaranya: Hewan (induk) beserta turunannya seperti Singa, Elang, Lebah, dan Paus menggunakan konsep Polymorphism serta menyimpan objek ke dalam database.



**Pembahasan**

Input

```
class Hewan():
    def __init__(self, id_jenis_hewan, nama_jenis_hewan, ciri_hewan):
        self.id_jenis_hewan = id_jenis_hewan
        self.nama_jenis_hewan = nama_jenis_hewan
        self.ciri_hewan = ciri_hewan

    def suara_hewan(self):
        return 'Suara Hewan'

    def berjalan_hewan(self):
        return 'Gerak Hewan'

class Singa(Hewan):
    def __init__(self, id_jenis_hewan, nama_jenis_hewan, ciri_hewan, id_singa, nama_singa):
        super().__init__( id_jenis_hewan, nama_jenis_hewan, ciri_hewan)
        self.id_singa = id_singa
        self.nama_singa = nama_singa

    def suara_hewan(self):
        return super().suara_hewan(), 'Roarrrr'

    def berjalan_hewan(self):
        return super().berjalan_hewan(), 'Bergerak dengan 4 Kaki'

    def bernafas_hewan(self):
        print('Singa bernafas menggunakan paru-paru')

class Elang(Hewan):
    def __init__(self, id_jenis_hewan, nama_jenis_hewan, ciri_hewan, id_elang, nama_elang):
        super().__init__( id_jenis_hewan, nama_jenis_hewan, ciri_hewan)
        self.id_elang = id_elang
        self.nama_elang = nama_elang
    def suara_hewan(self):
        return super().suara_hewan(), 'Eungiiiiiiiiik'

    def berjalan_hewan(self):
        return super().berjalan_hewan(), 'Bergerak dengan sayap dan 2 kaki'

    def bernafas_hewan(self):
        print('Elang bernafas menggunakan paru-paru')

class Lebah(Hewan):
    def __init__(self, id_jenis_hewan, nama_jenis_hewan, ciri_hewan, id_lebah, nama_lebah):
        super().__init__( id_jenis_hewan, nama_jenis_hewan, ciri_hewan)
        self.id_lebah = id_lebah
        self.nama_lebah = nama_lebah

    def suara_hewan(self):
        return super().suara_hewan(), 'Eeeezzzz'

    def berjalan_hewan(self):
        return super().berjalan_hewan(), 'Bergerak seperti angka 8'

    def bernafas_hewan(self):
        print('Lebah bernafas menggunakan trakea')

class Paus(Hewan):
    def __init__(self, id_jenis_hewan, nama_jenis_hewan, ciri_hewan, id_paus, nama_paus):
        super().__init__( id_jenis_hewan, nama_jenis_hewan, ciri_hewan)
        self.id_paus = id_paus
        self.nama_paus = nama_paus

    def suara_hewan(self):
        return super().suara_hewan(), 'ngaauuuuu ngggg'

    def berjalan_hewan(self):
        return super().berjalan_hewan(), 'Bergerak dengan sirip'

    def bernafas_hewan(self):
        return 'Paus bernafas menggunakan paru-paru'
```

Membuat kelas hewan sebagai superclass, dengan konstruktor yang berisi atribut id_jenis_hewan, nama_jenis_hewan, ciri_hewan dan memiliki 2 method yaitu suara_hewan dan berjalan_hewan.

kemudian membuat kelas singa yang merupakan subclass dari kelas hewan dengan konstruktornya menggunakan konstruktor dari superclass yaitu kelas Hewan dengan bantuan fungsi super() untuk memanggilnya. Selain menggunakan atribut yang sama dengan kelas parent, dalam kelas singa ini juga ditambahkan atribut baru yaitu id_singa dan nama_singa. Subclass singa ini mempunyai 3 method yang digunakan dengan dua diantaranya yaitu suara_hewan dan berjalan_hewan merupakan polimorfisme dari superclass Hewan dan menambahkan 1 method tambahan yaitu bernafas_hewan.

untuk subclass yang lainnya yaitu class Elang, class Lebah dan class Paus dilakukan dengan langkah yang sama seperti yang ada dalam subclass singa.

```
singa1 = Singa('Singa', 'Mamalia',
               'surai yang lebih menonjol dan lebih penuh yang menutupi seluruh kepala', 'L01', 'Singa
Afrika')
singa2 = Singa('Singa', 'Mamalia', 'Surai pendek gelap dan kurang berkembang', 'L02', 'Singa Asia')

dicts1 = singa1.__dict__
dicts2 = singa2.__dict__

def merge(s1, s2):
    sall = {}
    for k in s1:
        sall[k] = [s1[k], s2[k]]
    return sall

dictsall = merge(dicts1, dicts2)

for i, j in dictsall.items():
    print(i, ':', j)

print(singa1.suara_hewan())
print(singa1.berjalan_hewan())
print(singa1.bernafas_hewan())
```

Kemudian inisialisasi subclass singa yang sudah diberi value parameternya dengan variabel singa1. Kita juga membuat variabel singa2 untuk menginisialisasi subclass singa yang sudah diberi value parameternya namun dengan nilai valuenya yang berbeda karena singa memiliki jenis yang berbeda juga.

Kemudian simpan singa1 dan singa2 dalam bentuk dictionary dengan bantuan fungsi .__dict__, data singa yang teleh berbentuk dictionary ini akan ditampung oleh variabel dicts1 dan dicts2.

Agar dicts1 dan dicts2 dapat digabung menjadi sebuah dictionary kita membuat fungsi merge yang menstatemenkan: kita buat variabel sall adalah dictionary kosong, terdapat perulangan for jika k dalam s1 maka sall list ke k adalah gabungan list s1 yang berisi list k dan list s2 yang berisi list k, jika tidak maka akan mengembalikan nilai sall. Kemudian fungsi ini di tampung dalam variabel dictsall dan telah diberi value dimana s1nya adalah dicts1 dan s2nya adalah dicts2.

Agar memudahkan untuk melihat outputnya maka kita juga membuat perulangan for dimana untuk dan j dalam dictsall yang mempunyai items maka akan mencetak i : j, dimana i sebagai key dan j sebagai value.

Kemudian tidak lupa juga memanggil methode dari subclass yang telah dibuat yaitu dengan cara

print(singa1.suara_hewan())

print(singa1.berjalan_hewan())

print(singa1.bernafas_hewan())

untuk cara memberikan value dan menyimpan data dalam bentuk dictionary pada setiap classnya dilakukan dengan cara yang sama dengan cara yang dilakukan oleh subclass singa.

Output

```
id_jenis_hewan : ['Singa', 'Singa']
nama_jenis_hewan : ['Mamalia', 'Mamalia']
ciri_hewan : ['surai yang lebih menonjol dan lebih penuh yang menutupi seluruh kepala', 'Surai pendek
gelap dan kurang berkembang']
id_singa : ['L01', 'L02']
nama_singa : ['Singa Afrika', 'Singa Asia']
('Suara Hewan', 'Roarrrr')
('Gerak Hewan', 'Bergerak dengan 4 Kaki')
Singa bernafas menggunakan paru-paru
None
```

Input

```
elang1 = Elang('Elang', 'Unggas', 'Warna dominan adalah merah-coklat', 'E01', 'Elang Jawa')
elang2 = Elang('Elang', 'Unggas', 'Sayap bersayap membentuk huruf "C"', 'E02', 'Elang Ular')

dicte1 = elang1.__dict__
dicte2 = elang2.__dict__

def merge(e1, e2):
    eall = {}
    for k in e1:
        eall[k] = [e1[k], e2[k]]
    return eall

dicteall = merge(dicte1, dicte2)

for i, j in dicteall.items():
    print(i, ':', j)

print(elang1.suara_hewan())
print(elang1.berjalan_hewan())
print(elang1.bernafas_hewan())
```

Kemudian inisialisasi subclass singa yang sudah diberi value parameternya dengan variabel elang1. Kita juga membuat variabel elang2 untuk menginisialisasi subclass singa yang sudah diberi value parameternya namun dengan nilai valuenya yang berbeda karena elang memiliki jenis yang berbeda juga.

Kemudian simpan elang1 dan elang2 dalam bentuk dictionary dengan bantuan fungsi .__dict__, data lebah yang teleh berbentuk dictionary ini akan ditampung oleh variabel dicts1 dan dicts2.

Agar dicts1 dan dicts2 dapat digabung menjadi sebuah dictionary kita membuat fungsi merge yang menstatemenkan: kita buat variabel sall adalah dictionary kosong, terdapat perulangan for jika k dalam s1 maka sall list ke k adalah gabungan list s1 yang berisi list k dan list s2 yang berisi list k, jika tidak maka akan mengembalikan nilai sall. Kemudian fungsi ini di tampung dalam variabel dictsall dan telah diberi value dimana s1nya adalah dicts1 dan s2nya adalah dicts2.

Agar memudahkan untuk melihat outputnya maka kita juga membuat perulangan for dimana untuk dan j dalam dictsall yang mempunyai items maka akan mencetak i : j, dimana i sebagai key dan j sebagai value.

Kemudian tidak lupa juga memanggil methode dari subclass yang telah dibuat yaitu dengan cara

print(elang1.suara_hewan())

print(elang1.berjalan_hewan())

print(elang1.bernafas_hewan())

untuk cara memberikan value dan menyimpan data dalam bentuk dictionary pada setiap classnya dilakukan dengan cara yang sama dengan cara yang dilakukan oleh subclass elang .

Output

```
id_jenis_hewan : ['Elang', 'Elang']
nama_jenis_hewan : ['Unggas', 'Unggas']
ciri_hewan : ['Warna dominan adalah merah-coklat', 'Sayap bersayap membentuk huruf “C”']
id_elang : ['E01', 'E02']
nama_elang : ['Elang Jawa', 'Elang Ular']
('Suara Hewan', 'Eungiiiiiiiiik')
('Gerak Hewan', 'Bergerak dengan sayap dan 2 kaki')
Elang bernafas menggunakan paru-paru
None
```

Input

```
lebah1 = Lebah('Lebah', 'Serangga', 'warna kuning hitam', 'B01', 'Lebah Madu')
lebah2 = Lebah('Lebah', 'Serangga', 'warna hitam', 'B02', 'Lebah Tanah')

dictl1 = lebah1.__dict__
dictl2 = lebah2.__dict__

def merge(l1, l2):
    lall = {}
    for k in l1:
        lall[k] = [l1[k], l2[k]]
    return lall

dictlall = merge(dictl1, dictl2)

for i, j in dictlall.items():
    print(i, ':', j)

print(lebah1.suara_hewan())
print(lebah1.berjalan_hewan())
print(lebah1.bernafas_hewan())
```

Kemudian inisialisasi subclass singa yang sudah diberi value parameternya dengan variabel lebah1. Kita juga membuat variabel lebah2 untuk menginisialisasi subclass singa yang sudah diberi value parameternya namun dengan nilai valuenya yang berbeda karena singa memiliki jenis yang berbeda juga.

Kemudian simpan lebah1 dan lebah2 dalam bentuk dictionary dengan bantuan fungsi .__dict__, data lebah yang teleh berbentuk dictionary ini akan ditampung oleh variabel dicts1 dan dicts2.

Agar dicts1 dan dicts2 dapat digabung menjadi sebuah dictionary kita membuat fungsi merge yang menstatemenkan: kita buat variabel sall adalah dictionary kosong, terdapat perulangan for jika k dalam s1 maka sall list ke k adalah gabungan list s1 yang berisi list k dan list s2 yang berisi list k, jika tidak maka akan mengembalikan nilai sall. Kemudian fungsi ini di tampung dalam variabel dictsall dan telah diberi value dimana s1nya adalah dicts1 dan s2nya adalah dicts2.

Agar memudahkan untuk melihat outputnya maka kita juga membuat perulangan for dimana untuk dan j dalam dictsall yang mempunyai items maka akan mencetak i : j, dimana i sebagai key dan j sebagai value.

Kemudian tidak lupa juga memanggil methode dari subclass yang telah dibuat yaitu dengan cara

print(lebah1.suara_hewan())

print(lebah1.berjalan_hewan())

print(lebah1.bernafas_hewan())

untuk cara memberikan value dan menyimpan data dalam bentuk dictionary pada setiap classnya dilakukan dengan cara yang sama dengan cara yang dilakukan oleh subclass lebah

Output

```
id_jenis_hewan : ['Lebah', 'Lebah']
nama_jenis_hewan : ['Serangga', 'Serangga']
ciri_hewan : ['warna kuning hitam', 'warna hitam']
id_lebah : ['B01', 'B02']
nama_lebah : ['Lebah Madu', 'Lebah Tanah']
('Suara Hewan', 'Eeeezzzz')
('Gerak Hewan', 'Bergerak seperti angka 8')
Lebah bernafas menggunakan trakea
None
```

Input

```
paus1 = Paus('Paus', 'Mamalia', 'Badan besar dan lambat berenang', 'W01', 'Paus Biru')
paus2 = Paus('Paus', 'Mamalia', 'terdapat sebuah benjolan di atas kepalanya, sirip yang panjang',
'W02', 'Paus Bungkuk')

dictp1 = paus1.__dict__
dictp2 = paus2.__dict__

def merge(p1, p2):
    pall = {}
    for k in p1:
        pall[k] = [p1[k], p2[k]]
    return pall

dictpall = merge(dictp1, dictp2)

for i, j in dictpall.items():
    print(i, ':', j)

print(paus1.suara_hewan())
print(paus1.berjalan_hewan())
print(paus1.bernafas_hewan())
```

Kemudian inisialisasi subclass singa yang sudah diberi value parameternya dengan variabel paus1. Kita juga membuat variabel paus2 untuk menginisialisasi subclass singa yang sudah diberi value parameternya namun dengan nilai valuenya yang berbeda karena singa memiliki jenis yang berbeda juga.

Kemudian simpan paus1 dan paus2 dalam bentuk dictionary dengan bantuan fungsi .__dict__, data paus yang teleh berbentuk dictionary ini akan ditampung oleh variabel dicts1 dan dicts2.

Agar dicts1 dan dicts2 dapat digabung menjadi sebuah dictionary kita membuat fungsi merge yang menstatemenkan: kita buat variabel sall adalah dictionary kosong, terdapat perulangan for jika k dalam s1 maka sall list ke k adalah

gabungan list s1 yang berisi list k dan list s2 yang berisi list k, jika tidak maka akan mengembalikan nilai sall. Kemudian fungsi ini di tampung dalam variabel dictsall dan telah diberi value dimana s1nya adalah dicts1 dan s2nya adalah dicts2.

Agar memudahkan untuk melihat outputnya maka kita juga membuat perulangan for dimana untuk dan j dalam dictsall yang mempunyai items maka akan mencetak i : j, dimana i sebagai key dan j sebagai value.

Kemudian tidak lupa juga memanggil methode dari subclass yang telah dibuat yaitu dengan cara

print(paus1.suara_hewan())

print(paus1.berjalan_hewan())

print(paus1.bernafas_hewan())

untuk cara memberikan value dan menyimpan data dalam bentuk dictionary pada setiap classnya dilakukan dengan cara yang sama dengan cara yang dilakukan oleh subclass paus

Output

```
id_jenis_hewan : ['Paus', 'Paus']
nama_jenis_hewan : ['Mamalia', 'Mamalia']
ciri_hewan : ['Badan besar dan lambat berenang', 'terdapat sebuah benjolan di atas kepalanya, sirip
yang panjang']
id_paus : ['W01', 'W02']
nama_paus : ['Paus Biru', 'Paus Bungkuk']
('Suara Hewan', 'ngaauuuuu ngggg')
('Gerak Hewan', 'Bergerak dengan sirip')
Paus bernafas menggunakan paru-paru
```

# BAB V
# PENUTUP

## 5.1 Kesimpulan

Polimorfisme merupakan suatu konsep yang menyatakan sesuatu yang sama dapat memiliki berbagai bentuk dan perilaku berbeda. Polimorfisme merupakan kemampuan objek-objek yang berbeda kelas namun terkait dalam pewarisan untuk merespon secara berbeda terhadap suatu pesan yang sama.

## 5.2 Saran

Berdasarkan laporan yang telah dibuat, diharapkan agar kedepannya penulis dapatmembuat suatu program yang dapat menerapkan hasil dari fungsi yang telah dipelajari danmempelajari kembali mengenai fungsi yang dikira masih belum dipahami

# Daftar Pustaka

Adexe, A. (2021, Januari 30). *Belajar Object Oriented Program (OOP) dengan python*. Retrieved from anbidev: https://www.anbidev.com/python-oop-intro/#:~:text=Object%20Oriented%20Programming%20(OOP)%20adalah,dapat%20digunakan%20kembali%20(reuseable).

*Python Programming*. (n.d.). Retrieved from edureka!: https://www.edureka.co/blog/polymorphism-in-python/

Silvia. (2019, November 06). *Pengertian dan Konsep Dasar OOP*. Retrieved from Jetorbit: https://www.jetorbit.com/blog/pengertian-dan-konsep-dasar-object-oriented-programming-oop/