



MODUL PRAKTIKUM

POLYMORPHISM, DATA HIDING, OVERRIDING AND EXCEPTION HANDLING

I Gde Agung Sri Sidhimantra, S.Kom., M.Kom.

Binti Kholifah, S.Kom., M.Tr.Kom.

Moch Deny Pratama, S.Tr.Kom., M.Kom.

Dimas Novian Aditia Syahputra, S.Tr.T., M.Tr.T.



**MANAJEMEN
INFORMATIKA**

POLYMORPHISM, DATA HIDING, OVERRIDING AND EXCEPTION HANDLING

A. TUJUAN PEMBELAJARAN

1. Mahasiswa dapat mengimplementasikan **Polimorfisme** menggunakan metode dan fungsi umum.
2. Mahasiswa memahami konsep **Data Hiding** untuk melindungi atribut kelas.
3. Mahasiswa dapat memahami konsep **Overriding** metode dari kelas induk.
4. Mahasiswa memahami konsep **Exception Handling** untuk menangani kesalahan program.
5. Mahasiswa mampu mengimplementasikan konsep penerapan teori **Polymorphism, Data Hiding, Overriding and Exception Handling**.
6. Mahasiswa dapat **menganalisis hasil output** dari implementasi teori konsep ke dalam kode Python.

B. MATERI TEORI

1) POLYMORPHISM

Polimorfisme (Polymorphism) berasal dari bahasa Yunani yang berarti "banyak bentuk". Dalam konteks pemrograman berorientasi objek, polimorfisme memungkinkan satu metode atau fungsi memiliki perilaku yang berbeda tergantung pada objek atau data yang digunakan. Polimorfisme memungkinkan **satu metode atau fungsi memiliki perilaku berbeda** tergantung pada objek yang digunakan.

Jenis Polimorfisme dalam Python

- **Polimorfisme dengan Metode di Kelas**
Implementasi metode dengan nama yang sama tetapi perilaku berbeda di kelas yang berbeda.
- **Polimorfisme dengan Pewarisan (Inheritance)**
Subkelas dapat mengubah (override) metode kelas induk untuk memberikan perilaku spesifik.
- **Polimorfisme dengan Fungsi Umum**

Fungsi generik dapat bekerja dengan berbagai objek asalkan objek memiliki metode atau atribut yang sesuai.

a) Polimorfisme dengan Metode di Kelas

Nama metode yang sama digunakan dalam berbagai kelas, tetapi implementasinya berbeda.

```
class Dog:
    def sound(self):
        return "Woof"

class Cat:
    def sound(self):
        return "Meow"

# Polymorphism example
animals = [Dog(), Cat()]
for animal in animals:
    print(animal.sound())
```

Output:

```
Woof
Meow
```

Polimorfisme di sini memungkinkan kita menggunakan metode `sound` tanpa peduli apakah objek tersebut adalah `Dog` atau `Cat`. Selama objek memiliki metode `sound`, iterasi dapat berjalan dengan baik, dan metode tersebut akan menghasilkan perilaku yang sesuai dengan kelas masing-masing. Keuntungan dari pendekatan ini yaitu tidak perlu menulis logika terpisah untuk setiap jenis hewan, dan ketika menambahkan kelas baru, hanya perlu implementasi metode `sound` tanpa mengubah kode iterasi.

b) Polimorfisme dengan Pewarisan

Subkelas dapat mengubah perilaku metode kelas induk melalui overriding.

```
class Animal:
    def sound(self):
        return "Some sound"

class Dog(Animal):
    def sound(self):
        return "Woof"

class Cat(Animal):
    def sound(self):
        return "Meow"

# Polymorphism example
animals = [Animal(), Dog(), Cat()]
for animal in animals:
    print(animal.sound())
```

Output:

```
Some sound
Woof
Meow
```

Kelas Animal adalah kelas induk (parent class). Kelas ini memiliki metode sound yang mengembalikan string "Some sound". Metode ini akan digunakan sebagai "default" untuk semua kelas turunan kecuali jika metode tersebut di-override. Dog dan Cat adalah kelas turunan (subclass) dari Animal. Metode sound pada Dog dan Cat meng-override metode sound milik Animal, sehingga memunculkan suara "Woof" dan "Meow". Sementara list animals dibuat yang berisi objek dari ketiga kelas: Animal, Dog, dan Cat. List ini menunjukkan polimorfisme, karena meskipun semua objek berasal dari kelas yang berbeda, mereka memiliki metode yang sama (sound) yang dipanggil dengan cara yang sama.

Dengan melakukan iterasi pada daftar animals, objek dari setiap kelas diproses satu per satu. Metode sound dari objek tersebut dipanggil menggunakan animal.sound().

- Iterasi Pertama: Objek adalah Animal(), sehingga animal.sound() mengembalikan "Some sound".
- Iterasi Kedua: Objek adalah Dog(), sehingga animal.sound() mengembalikan "Woof".
- Iterasi Ketiga: Objek adalah Cat(), sehingga animal.sound() mengembalikan "Meow".

c) Polimorfisme dengan Fungsi Umum

Fungsi dapat bekerja dengan objek dari berbagai kelas selama mereka memiliki metode yang diperlukan.

```
class Bird:
    def fly(self):
        return "Flapping wings"

class Airplane:
    def fly(self):
        return "Jet engine roaring"

def make_fly(entity):
    print(entity.fly())

# Polymorphism example
make_fly(Bird())      # Output: Flapping wings
make_fly(Airplane()) # Output: Jet engine roaring
```

Output:

```
Flapping wings
Jet engine roaring
```

Kelas Bird memiliki metode fly yang mengembalikan string "Flapping wings", merepresentasikan cara burung terbang, yaitu dengan mengepakkan sayap. Fungsi make_fly adalah fungsi generik yang dapat menerima parameter apa pun. Begitupun dengan kelas Airplane. Dalam fungsi make_fly, metode fly dari parameter yang diterima dipanggil menggunakan entity.fly(). Objek yang diterima oleh make_fly harus memiliki metode fly. Jika tidak, akan terjadi error AttributeError. Saat make_fly dipanggil, ketika parameter adalah objek Bird, metode fly dari kelas Bird dipanggil, sehingga menghasilkan output "Flapping wings".

Sama halnya dengan ketika parameter adalah objek Airplane. Fungsi make_fly dapat bekerja dengan objek dari berbagai kelas yang berbeda (Bird dan Airplane), asalkan mereka memiliki metode fly. Perilaku fungsi berubah sesuai dengan implementasi metode fly pada masing-masing kelas. Dengan menggunakan fungsi generik seperti make_fly, kita tidak perlu menulis kode khusus untuk setiap jenis objek (Bird, Airplane, dll.). Selama objek memiliki metode yang diharapkan, fungsi dapat bekerja tanpa masalah.

2) DATA HIDING

Data Hiding adalah prinsip dalam OOP yang digunakan untuk membatasi akses ke atribut atau metode tertentu dalam sebuah kelas, biasanya dengan menjadikannya privat. Di Python, atribut atau metode yang ingin disembunyikan menggunakan dua garis bawah (__) di awal nama. *Data hiding* merupakan **aspek khusus dari enkapsulasi** yang berfokus pada pembatasan akses langsung ke atribut atau metode dari luar kelas. Atribut atau metode yang disembunyikan biasanya diberi tanda dengan awalan **dua garis bawah (__)** dalam Python. Tujuan utama *data hiding* adalah.

1. Melindungi data dari modifikasi yang tidak diinginkan.
2. Memberikan kontrol penuh atas data melalui antarmuka tertentu.

Contoh Konsep

- Atribut privat tidak dapat diakses langsung dari luar kelas.
- Hanya metode tertentu (seperti *getter* dan *setter*) yang dapat mengakses data tersebut.

Perbedaan Utama

Aspek	Enkapsulasi	Data Hiding
Fokus	Menggabungkan data dan metode ke dalam satu unit (kelas).	Menyembunyikan atribut atau metode dari akses langsung.
Tujuan	Meningkatkan modularitas dan organisasi kode.	Melindungi data dari modifikasi yang tidak diinginkan.

Aspek	Enkapsulasi	Data Hiding
Aksesibilitas	Bisa mencakup atribut dan metode publik, privat, atau proteksi.	Biasanya hanya fokus pada atribut atau metode privat.
Penggunaan	Memerlukan metode publik untuk antarmuka kelas.	Memerlukan <i>getter</i> dan <i>setter</i> untuk akses data privat.

Data hiding adalah teknik untuk melindungi atribut kelas dari akses langsung, sering digunakan untuk menjaga keamanan dan integritas data. Atribut yang disembunyikan diberi awalan `__` (contoh: `__balance`).

Langkah Praktikum

1. Buat **Class BankAccount** dengan atribut private `_balance`.
2. Tambahkan metode **getter `get_balance()`** untuk mengakses saldo.
3. Tambahkan metode **setter `deposit()`** untuk menambah saldo dengan validasi data.

```
class BankAccount:
    def __init__(self):
        self.__balance = 0 # Private attribute

    def get_balance(self):
        return self.__balance

    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
        else:
            print("Invalid deposit amount!")

# Implementasi
account = BankAccount()
account.deposit(1000)
print("Balance:", account.get_balance())
account.deposit(-500)
```

Hasil Output:

Balance: 1000

Invalid deposit amount!

Tujuan Kode:

Memahami penggunaan data hiding → Atribut saldo hanya dapat diakses atau diubah melalui metode tertentu sehingga data lebih aman dari manipulasi langsung.

3) OVERRIDING

Overriding adalah konsep dalam pemrograman berorientasi objek di mana sebuah metode di subkelas menggantikan implementasi metode dengan nama yang sama dari kelas induknya. Overriding mendukung **polimorfisme**, memungkinkan objek dari kelas turunan berperilaku berbeda meskipun menggunakan metode yang sama. **Overriding** merupakan kemampuan subkelas untuk mengganti metode dari kelas induk. Ini mendukung polimorfisme dan memungkinkan perilaku unik untuk setiap subkelas.

Langkah Praktikum

1. Buat parent class Vehicle dengan metode move().
2. Buat subclass Car dan Bike, masing-masing meng-override metode move().
3. Tulis contoh implementasi untuk memanggil metode move() dari objek Car dan Bike.

```
class Vehicle:
    def move(self):
        return "Vehicle is moving"

class Car(Vehicle):
    def move(self):
        return "Car is driving"

class Bike(Vehicle):
    def move(self):
        return "Bike is riding"

# Implementasi
vehicle = Vehicle()
car = Car()
bike = Bike()

print(vehicle.move())
print(car.move())
print(bike.move())
```

Hasil Output:

Vehicle is moving

Car is driving

Bike is riding

Tujuan Kode:

Mengimplementasikan overriding → Subclass dapat memberikan implementasi spesifik untuk metode kelas induk.

4) EXCEPTION HANDLING

Exception handling adalah mekanisme untuk **menangani kesalahan** atau pengecualian yang terjadi saat program berjalan. Dalam Python, kita menggunakan blok try, except, else, dan finally untuk menangani pengecualian.

Pentingnya Exception Handling.

1. **Menangkap Kesalahan** → Menghindari program berhenti mendadak karena kesalahan.
2. **Meningkatkan Keamanan Data** → Melindungi data dari kerusakan akibat kesalahan.
3. **Memberikan Informasi kepada Pengguna** → Mengarahkan pengguna dengan pesan yang sesuai.

Langkah Praktikum

1. Buat fungsi pembagi bernama **divide_numbers()** yang menerima dua parameter.
2. Tambahkan mekanisme untuk menangkap error seperti **ZeroDivisionError** dan **TypeError**.
3. Tulis contoh implementasi dengan input yang valid dan error.

```
def divide_numbers(a, b):  
    try:  
        result = a / b  
        return f"Result: {result}"  
    except ZeroDivisionError:  
        return "Error: Division by zero is not allowed."  
    except TypeError:  
        return "Error: Both inputs must be numbers."  
  
# Implementasi  
print(divide_numbers(10, 2))  
print(divide_numbers(10, 0))  
print(divide_numbers(10, 'a'))
```

Hasil Output:

Result: 5.0

Error: Division by zero is not allowed.

Error: Both inputs must be numbers.

Tujuan Kode:

Memahami exception handling → Error yang terjadi selama runtime dapat ditangani tanpa menghentikan program.

C. LANGKAH-LANGKAH PRAKTIKUM

1. POLYMORPHISM

Contoh pertama adalah program menggunakan Tkinter yang mengimplementasikan **Polimorfisme**. Program ini akan menampilkan jendela dengan tombol yang, ketika diklik, menghasilkan suara berbeda berdasarkan jenis hewan (burung dan anjing). Polimorfisme diterapkan dalam cara suara dihasilkan, tergantung pada objek yang dipilih.

Langkah 1: Install Python dan Tkinter jika belum terpasang. Tkinter sudah termasuk dalam distribusi standar Python.

```
1 import tkinter as tk
```

Langkah 2: Buat kelas Animal, Bird, dan Dog dengan metode make_sound(). Kelas Bird dan Dog meng-override metode make_sound dari kelas Animal.

```
3 # Kelas induk Animal
4 class Animal:
5     def make_sound(self):
6         return "Some sound"
7
8 # Kelas turunan Bird
9 class Bird(Animal):
10     def make_sound(self):
11         return "Tweet tweet"
12
13 # Kelas turunan Dog
14 class Dog(Animal):
15     def make_sound(self):
16         return "Woof woof"
17
18 # Fungsi untuk menampilkan suara berdasarkan jenis hewan yang dipilih
19 def show_sound(animal):
20     label_result.config(text=animal.make_sound())
```

Fungsi show_sound menerima objek animal dan memanggil metode make_sound() dari objek tersebut. Ini menunjukkan polimorfisme, karena metode yang dipanggil berbeda-beda tergantung jenis objek yang diteruskan.

Langkah 3: Buat jendela Tkinter dengan tombol untuk memilih jenis hewan. Setiap tombol akan memanggil metode make_sound sesuai dengan objek yang dipilih.

```
22 # Membangun jendela utama menggunakan Tkinter
23 root = tk.Tk()
24 root.title("Polimorfisme di Tkinter")
25
26 # Label untuk menampilkan hasil suara
27 label_result = tk.Label(root, text="Klik salah satu tombol untuk mendengar suara hewan.", font=("Arial", 14))
28 label_result.pack(pady=20)
```

Jendela Tkinter dibuat dengan judul "Polimorfisme di Tkinter".

```

30 # Tombol untuk memilih Burung
31 button_bird = tk.Button(root, text="Burung", font=("Arial", 12), command=lambda: show_sound(Bird()))
32 button_bird.pack(pady=10)
33
34 # Tombol untuk memilih Anjing
35 button_dog = tk.Button(root, text="Anjing", font=("Arial", 12), command=lambda: show_sound(Dog()))
36 button_dog.pack(pady=10)
37
38 # Tombol untuk memilih Hewan Umum
39 button_animal = tk.Button(root, text="Hewan Umum", font=("Arial", 12), command=lambda: show_sound(Animal()))
40 button_animal.pack(pady=10)
41
42 # Menjalankan aplikasi Tkinter
43 root.mainloop()

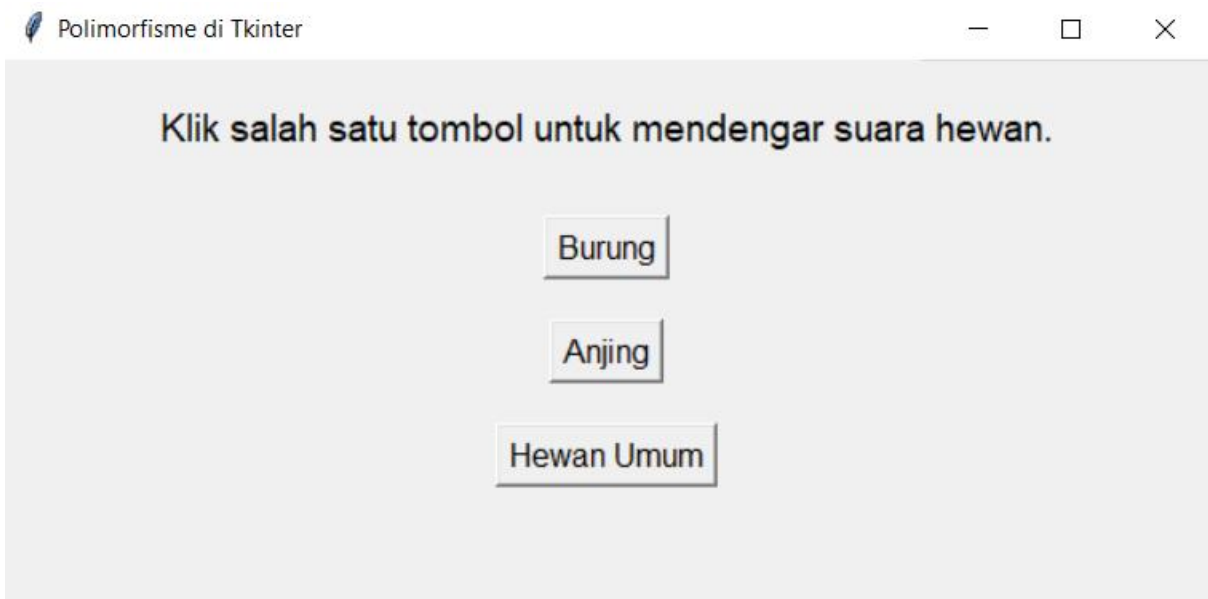
```

Hasil Output:

Tiga tombol disediakan untuk memilih hewan yaitu :

- Tombol untuk Burung memanggil objek Bird.
- Tombol untuk Anjing memanggil objek Dog.
- Tombol untuk Hewan Umum memanggil objek Animal sebagai default.

Ketika tombol diklik, metode `show_sound()` dipanggil dan menampilkan suara sesuai dengan objek yang dipilih pada label `label_result`.



2. DATA HIDING

Langkah 4: Persiapan Library

Pastikan Python dan pustaka **Tkinter** sudah terinstal. Tkinter termasuk dalam distribusi standar Python.

```

import tkinter as tk
from tkinter import messagebox

```

Langkah 5: Implementasi Data Hiding

1. Buat kelas BankAccount dengan atribut privat `__balance`.

```
class BankAccount:
    def __init__(self):
        self.__balance = 0 # Private attribute

    def get_balance(self):
        return self.__balance

    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
        else:
            raise ValueError("Invalid deposit amount!")
```

2. Tambahkan metode untuk mengakses saldo (`get_balance`) dan menambah saldo (`deposit`).

```
def add_balance():
    try:
        amount = int(entry_amount.get())
        account.deposit(amount)
        label_balance.config(text=f"Balance: {account.get_balance()}")
    except ValueError as e:
        messagebox.showerror("Error", str(e))
```

3. Tampilkan saldo awal di label **Tkinter**, dan tambahkan tombol untuk melakukan deposit.

```
def add_balance():
    try:
        amount = int(entry_amount.get())
        account.deposit(amount)
        label_balance.config(text=f"Balance: {account.get_balance()}")
    except ValueError as e:
        messagebox.showerror("Error", str(e))

# GUI Tkinter
root = tk.Tk()
root.title("Data Hiding in BankAccount")

account = BankAccount()

label_balance = tk.Label(root, text=f"Balance: {account.get_balance()}")
label_balance.pack()

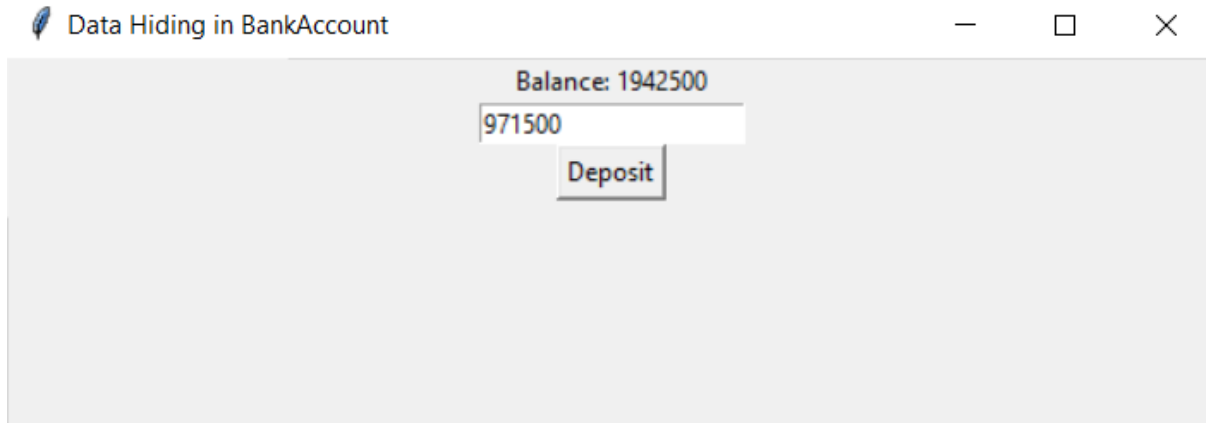
entry_amount = tk.Entry(root)
entry_amount.pack()

button_deposit = tk.Button(root, text="Deposit", command=add_balance)
button_deposit.pack()

root.mainloop()
```

Hasil Output:

- Tampilan awal menunjukkan saldo = 0.
- Jika pengguna memasukkan nilai valid, saldo bertambah. Jika tidak valid, muncul pesan error.



3. OVERRIDING

Langkah 6: Implementasi Overriding

1. Buat parent class Shape dengan metode area() dan perimeter().

```
import tkinter as tk
from math import pi

class Shape:
    def area(self):
        return "Not implemented"

    def perimeter(self):
        return "Not implemented"
```

2. Buat subclass Rectangle dan Circle yang meng-override metode tersebut.

```
class Rectangle(Shape):
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

    def perimeter(self):
        return 2 * (self.length + self.width)
```

```
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return pi * self.radius ** 2

    def perimeter(self):
        return 2 * pi * self.radius
```

3. Buat Function Calculate di luar kedua Class tersebut, tampilkan hasil area dan boundary berdasarkan input di GUI Tkinter.

```
def calculate():
    try:
        shape_type = shape_var.get()
        if shape_type == "Rectangle":
            shape = Rectangle(int(entry_param1.get()), int(entry_param2.get()))
        elif shape_type == "Circle":
            shape = Circle(int(entry_param1.get()))
        else:
            raise ValueError("Invalid shape")

        label_result.config(text=f"Area: {shape.area()}, Perimeter: {shape.perimeter()}")
    except ValueError as e:
        messagebox.showerror("Error", str(e))
```

```
root = tk.Tk()
root.title("Overriding in Shapes")

shape_var = tk.StringVar(value="Rectangle")

tk.Radiobutton(root, text="Rectangle", variable=shape_var, value="Rectangle").pack()
tk.Radiobutton(root, text="Circle", variable=shape_var, value="Circle").pack()

tk.Label(root, text="Parameter 1:").pack()
entry_param1 = tk.Entry(root)
entry_param1.pack()

tk.Label(root, text="Parameter 2 (if Rectangle):").pack()
entry_param2 = tk.Entry(root)
entry_param2.pack()

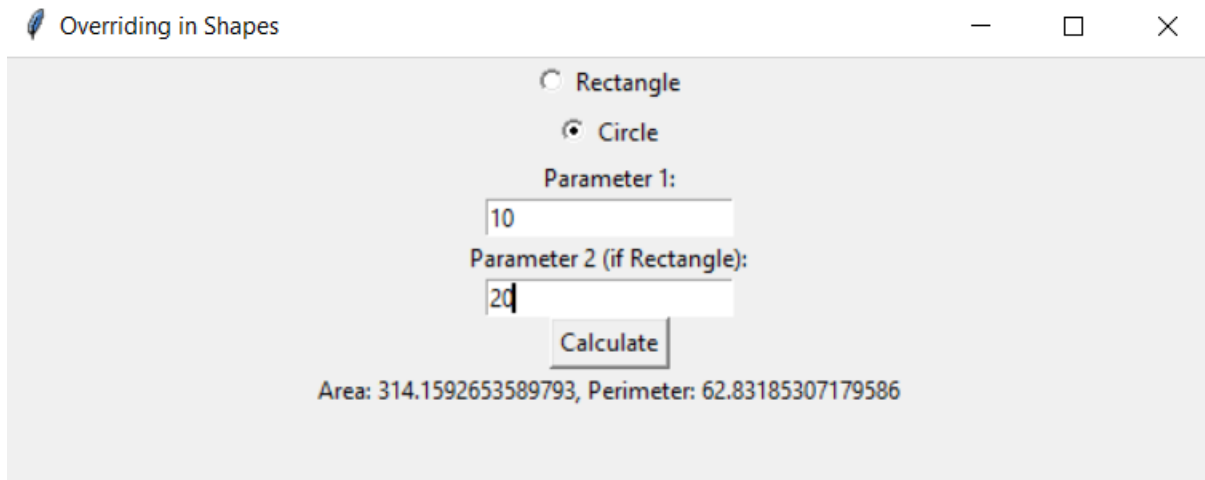
button_calculate = tk.Button(root, text="Calculate", command=calculate)
button_calculate.pack()

label_result = tk.Label(root, text="Area: , Perimeter: ")
label_result.pack()

root.mainloop()
```

Hasil Output:

- Untuk Rectangle, pengguna memasukkan panjang dan lebar.
- Untuk Circle, pengguna hanya memasukkan jari-jari.
- Output menunjukkan area dan perimeter.



4. EXCEPTION HANDLING

Langkah 7: Implementasi Exception Handling

1. Buat fungsi pembagian (divide_numbers) dengan penanganan error (Exception Handling).

```
import tkinter as tk
from tkinter import messagebox

def divide_numbers():
    try:
        num1 = int(entry_num1.get())
        num2 = int(entry_num2.get())
        result = num1 / num2
        label_result.config(text=f"Result: {result}")
    except ZeroDivisionError:
        messagebox.showerror("Error", "Division by zero is not allowed.")
    except ValueError:
        messagebox.showerror("Error", "Please enter valid numbers.")
```

2. Tambahkan GUI untuk memasukkan dua angka dan tombol untuk memproses pembagian.

```
root = tk.Tk()
root.title("Exception Handling in Division")

tk.Label(root, text="Number 1:").pack()
entry_num1 = tk.Entry(root)
entry_num1.pack()

tk.Label(root, text="Number 2:").pack()
entry_num2 = tk.Entry(root)
entry_num2.pack()

button_divide = tk.Button(root, text="Divide", command=divide_numbers)
button_divide.pack()

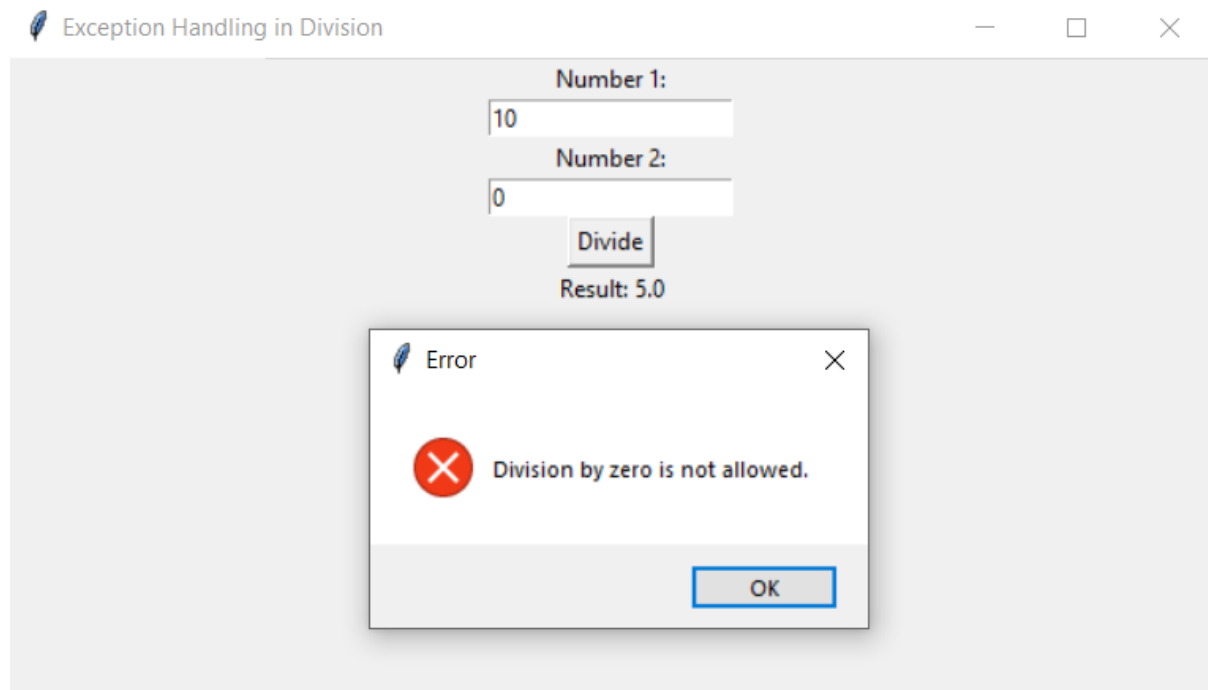
label_result = tk.Label(root, text="Result: ")
label_result.pack()

root.mainloop()
```

3. Tampilkan error jika ada pembagian dengan nol atau input bukan angka.

Hasil Output:

- Jika pembagian valid, hasil ditampilkan.
- Jika terjadi kesalahan (pembagian dengan nol atau input salah), muncul pesan error.



KESIMPULAN PRAKTIKUM

- **Polymorphism** memberikan fleksibilitas dalam penggunaan metode pada objek yang berbeda.
- **Data Hiding** menjaga keamanan dan integritas data.
- **Overriding** memungkinkan subclass untuk memberikan perilaku khusus.
- **Exception Handling** memastikan program tetap berjalan meskipun terjadi kesalahan.

D. TUGAS PRAKTIKUM

1. Berdasarkan Langkah-Langkah Praktikum **Nomor 1**, Tambahkan **2 Button** untuk menampilkan Suara dari 2 Objek Hewan yang berbeda.
2. Berdasarkan Langkah-Langkah Praktikum **Nomor 5 - 7**, Tambahkan tombol untuk **Reset Input** di setiap bagian Teori Konsep Praktikum.