

**NAME: EULITA MUBAIWA**

**REG NUMBER**: **PG247122H**

**YEAR: 1.2**

**ASSIGNMENT: 1**

**LECTURE: MS MURIMI**

1. # Subclass Car

class Car(Vehicle):

def \_\_init\_\_(self, brand, speed, doors):

super().\_\_init\_\_ (brand, speed)

self.doors = doors

# Overriding the describe method

def describe(self):

return f"This is a car made by {self.brand}. It has {self.doors} doors and a top speed of {self.speed} km/h."

# Subclass Bike

class Bike(Vehicle):

def \_\_init\_\_(self, brand, speed, bike\_type):

super().\_\_init\_\_ (brand, speed)

self.bike\_type = bike\_type

# Overriding the describe method

def describe(self):

return f"This is a {self.bike\_type} bike made by {self.brand} with a top speed of {self.speed} km/h."

# Example usage

my\_car = Car("Toyota", 180, 4)

my\_bike = Bike("Yamaha", 120, "sports")

print(my\_car.describe()) # Output: This is a car made by Toyota. It has 4 doors and a top speed of 180 km/h.

print(my\_bike.describe()) # Output: This is a sports bike made by Yamaha with a top speed of 120 km/h.

1. # Base class

class Shape:

def area(self):

raise NotImplementedError("Subclasses must implement this method")

# Circle subclass

class Circle(Shape):

def \_\_init\_\_(self, radius):

self.radius = radius

def area(self):

return math.pi \* self.radius \*\* 2

# Rectangle subclass

class Rectangle(Shape):

def \_\_init\_\_(self, width, height):

self.width = width

self.height = height

def area(self):

return self.width \* self.height

# Function to calculate total area

def calculate\_total\_area(shapes):

total\_area = 0

for shape in shapes:

total\_area += shape.area()

# Polymorphism in action

return total\_area

# Example usage

circle1 = Circle(5) # Circle with radius 5

rectangle1 = Rectangle(4, 6) # Rectangle with width 4 and height 6

shapes\_list = [circle1, rectangle1]

total\_area = calculate\_total\_area(shapes\_list)

print(f"Total area of all shapes: {total\_area:.2f}")

# Base class

class Shape:

def \_\_init\_\_(self, name):

self.name = name # Initialization logic in the base class

def calculate\_area(self):

raise NotImplementedError("Subclasses must implement the calculate\_area method!")

1. # Derived class

class Rectangle(Shape):

def \_\_init\_\_(self, name, width, height):

# Call the base class constructor

super().\_\_init\_\_(name)

self.width = width

self.height = height

# Override the calculate\_area method

def calculate\_area(self):

return self.width \* self.height

# Example usage

rect = Rectangle("MyRectangle", 4, 5)

print(f"{rect.name} area: {rect.calculate\_area()}") # Output: MyRectangle area: 20

# Class for Dog

class Dog:

def make\_sound(self):

return "Woof! Woof!"

# Class for Cat

class Cat:

def make\_sound(self):

return "Meow! Meow!"

# Function to process sound

def process\_sound(sound\_object):

print(sound\_object.make\_sound())

# Example usage

dog = Dog()

cat = Cat()

process sound(dog) # Output: Woof! Woof!

process\_sound(cat) # Output: Meow! Meow!

from abc import ABC, abstractmethod

1. # Define an abstract base class

class FileHandler(ABC):

@abstractmethod

def read(self):

pass

@abstractmethod

def write(self, data):

pass

1. # Implement specific file handlers

Class TextFileHandler (File Handler):

def read(self):

print ("Reading from a text file")

return "Text file content"

def write(self, data):

print(f"Writing to a text file: {data}")

Class BinaryFileHandler (FileHandler):

def read(self):

print("Reading from a binary file")

return b"Binary file content"

def. write (self, data):

print (“Writing to a binary file: {data}")

# Usage example

text handler = TextFileHandler ()

text\_handler.read ()

text\_handler.write ("Some data")

binary handler = BinaryFileHandler ()

binary\_handler.read ()

binary\_handler.write (b"Some binary data")