1.Write a python program to create a base class "Shape" with methods to calculate area and perimeter. Then, create derived classes "Circle" and "Rectangle" that inherit from the base class and calculate their respective areas and perimeters. Demonstrate their usage in a program. You are developing an online quiz application where users can take quizzes on various topics and receive scores.

1. Createa class for quizzes and questions.
2. Implement a scoring system that calculates the user's score on a qu iz
3. How would you store and retrieve user progress,on a quiz.including quiz history and scores?

In [3]:
```
import math
#Base class
class Shape:
    def calculate_area(self):
        pass

    def calculate_perimeter(self):
        pass
#Derived class
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def calculate_area(self):
        return math.pi * self.radius**2

    def calculate_perimeter(self):
        return 2 * math.pi * self.radius

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

    def calculate_area(self):
        return self.width * self.height

    def calculate_perimeter(self):
        return 2 * (self.width + self.height)


circle = Circle(2)
rectangle = Rectangle(4, 8)



print("Area of rectangle :", rectangle.calculate_area())
print("Perimeter of rectangle:", rectangle.calculate_perimeter())
print()
print()
print("Area of the circle :", circle.calculate_area())
print("Perimeter of the circle :", circle.calculate_perimeter())
```

```
Area of rectangle : 32
Perimeter of rectangle: 24


Area of the circle : 12.566370614359172
Perimeter of the circle : 12.566370614359172
```

In [4]:
```
class Question:
    def __init__(self, text, options, correct_option):
        self.text = text
        self.options = options
        self.correct_option = correct_option

class Quiz:
    def __init__(self, name):
        self.name = name
        self.questions = []

    def add_question(self, question):
        self.questions.append(question)

    def take_quiz(self):
        score = 0
        for question in self.questions:
            print(question.text)
            for i, option in enumerate(question.options, start=1):
                print(f"{i}. {option}")
            user_answer = int(input("Your answer (enter the number of the corre
            if user_answer == question.correct_option:
                score += 1
        return score


question1 = Question("What is the area of rectangle with radius 2 ?", ["32", "3
question2 = Question("What is the formulae for area of rectangle ?", ["(w*h)",

quiz = Quiz("Quiz on areas and perimeters")
quiz.add_question(question1)
quiz.add_question(question2)

user_score = quiz.take_quiz()
print(f"Your score: {user_score}")
```

```
What is the area of rectangle with radius 2 ?
1. 32
2. 34
3. 36
Your answer (enter the number of the correct option): 1
What is the formulae for area of rectangle ?
1. (w*h)
2. 2*(w+h)
3. none
Your answer (enter the number of the correct option): 1
Your score: 2
```

2.Write a python script to create a class "Person" with private attributes for age and name. Implement a method to calculate a person's eligibility for voting based on their

age. Ensure that age cannot be accessed directly but only through a getter method.

In [18]:
```
class Person:
    def __init__(self, name, age):
        self.__name = name # Private attribute for name
        self.__age = age # Private attribute for age

    def get_name(self):
        return self.__name

    def get_age(self):
        return self.__age

    def eligible(self):
        return self.__age >= 19


p = Person("Shaik Tauheer Ahamed ", 23)
print(f"{p.get_name()} is {p.get_age()} years old.")
if p.eligible():
    print(f"{p.get_name()} is eligible to vote. He is major")
else:
    print(f"{p.get_name()} is not eligible to vote.He is minor")
```

```
Shaik Tauheer Ahamed  is 23 years old.
Shaik Tauheer Ahamed  is eligible to vote. He is major
```

3.You are tasked with designing a Python class hierarchy for a simple banking system. The system should be able to handle different types of accounts, such as Savings Accounts and Checking Accounts. Both account types should have common attributes like an account number, account holder's name, and balance. However, Savings Accounts should have an additional attribute for interest rate, while Checking Accounts should have an attribute for overdraft limit.

1. Create a Python class called Bank Account with the following attributes and methods:

    a. Attributes: account_number, account holder_name, balance

    b. Methods: init 0 (constructor), deposit(), and withdraw

2. Create two subclasses, Savings Account and CheckingAccount, that inherit from the BankAccount class.

3. Add the following attributes and methods to each subclass:

    a. SavingsAccount:

        i. Additional attribute: interest_rate

        ii. Method: calculate_interest(), which calculates and adds interest to the account based on the interest rate.

    b. Checking Account:

        i. Additional attribute: overdraft limit

        ii. Method: withdraw(), which allows withdrawing money up to the overdraft limit (if available) without additional fees.

4. Write a program that creates instances of both Savings Account and Checking Account and demonstrates the use of their methods

5.Implement proper encapsulation by making the attributes private where necessary and providing getter and setter methods as needed.

6.Handle any potential errors or exceptions that may occur during operations like withdrawals, deposits, or interest calculations

In [1]:
```
class BankAccount:
    def __init__(self, acc_num, name, bal):
        self.acc_num = acc_num
        self.name = name
        self.bal = bal

    def deposit(self, depo_amount):
        if(depo_amount > 0):
            self.bal += depo_amount
            print(f"Deposited amoont is {depo_amount} and the Balance is: {self
        else:
            raise ValueError(f"Enter Amount is less than zero\n")
    def withdraw(self, wd_amount):
        if(self.bal >= wd_amount and wd_amount>0):
            self.bal -= wd_amount
            print(f"Withdraw amoont is {wd_amount} and the Balance is: {self.ba
        else:
            raise ValueError(f"Insufficient Balance\n")

class SavingsAccount(BankAccount):
    def __init__(self, interest, acc_num, name, bal):
        self.interest = interest
        super().__init__(acc_num, name, bal)
    def calculateInterest(self):
        interest_yearly = (self.bal*1*self.interest)
        print(f"\nInterest on savings account of is: {interest_yearly}\n")
        super().deposit(interest_yearly)


class CheckingAccount(BankAccount):
    def __init__(self, overdraft_limit, acc_num, name, bal):
        self.odlimit = overdraft_limit
        super().__init__(acc_num, name, bal)

    def withdraw(self, wd_amount):
        if(wd_amount > 0 and (self.bal+self.odlimit) >= wd_amount):
            super().withdraw(wd_amount)
            print(f"With Draw amount {wd_amount} from and the remaining balance
        else:
            raise ValueError(f"Entered Amount exceded the over drawft limit\n")



if __name__ == "__main__":
    p1name = input("Enter customer1 name: \n")
    p1acc_num = input("Enter customer1 account numnber: \n")
    p1bal = float(input("Enter Balance1: \n"))

    p2name = input("Enter customer2 name: \n")
    p2acc_num = input("Enter customer2 account numnber: \n")
    p2bal = float(input("Enter Balance2: \n"))

    bnk_acc1 = BankAccount(p1acc_num, p1name, p1bal)
    bnk_acc2 = BankAccount(p2acc_num, p2name, p2bal)

    psint = (int(input("Enter interest rate for savings account: \n"))/100)
    sav_acc = SavingsAccount(psint, p1acc_num, p1name, p1bal)
```

```
pCodlimit = int(input("Enter overdrawft limit: \n"))
check_acc = CheckingAccount(pCodlimit, p2acc_num, p2name, p2bal)


sav_acc.calculateInterest()

depo_sav = float(input("Enter deposit amount from savings account: \n"))
sav_acc.deposit(depo_sav)

wd_sav = float(input("Enter withdraw amount from savings account: \n"))
sav_acc.withdraw(wd_sav)

depo_check = float(input("Enter deposit amount from checking account: \n"))
check_acc.deposit(depo_check)

wd_check = float(input("Enter withdraw amount from checking account: \n"))
check_acc.withdraw(wd_check)
```

```
Enter customer1 name:
tauheer
Enter customer1 account numnber:
4545454
Enter Balance1:
450
Enter customer2 name:
mastan vali
Enter customer2 account numnber:
584554
Enter Balance2:
254
Enter interest rate for savings account:
554
Enter overdrawft limit:
78

Interest on savings account of is: 2493.0

Deposited amoont is 2493.0 and the Balance is: 2943.0

Enter deposit amount from savings account:
78
Deposited amoont is 78.0 and the Balance is: 3021.0

Enter withdraw amount from savings account:
5
Withdraw amoont is 5.0 and the Balance is: 3016.0

Enter deposit amount from checking account:
4
Deposited amoont is 4.0 and the Balance is: 258.0

Enter withdraw amount from checking account:
4
Withdraw amoont is 4.0 and the Balance is: 254.0

With Draw amount 4.0 from and the remaining balance is: 254.0
```

4.You are developing an employee management system for a company. Ensure that the system utilizes encapsulation and polymorphism to handle different types of employees, such as full-time and part-time employees.

1. Create a base class called "Employee" with private attributes for name, employeeID, and salary. Implement getter and setter methods for these attributes.
2. Design two subclasses, "FullTimeEmployee" and "PartTimeEmployee," that inherit from "Employee." These subclasses should encapsulate specific properties like hours worked (for part-time employees) and annual salary (for full-time employees).

In [22]:
```
# Base class Employee
class Employee:
    def __init__(self, name, employeeID, salary):
        self.__name = name
        self.__employeeID = employeeID
        self.__salary = salary
    def get_name(self):
        return self.__name
    def set_name(self, name):
        self.__name = name
    def get_employeeID(self):
        return self.__employeeID
    def set_employeeID(self, employeeID):
        self.__employeeID = employeeID
    def get_salary(self):
        return self.__salary
    def set_salary(self, salary):
        self.__salary = salary
    # Salary calculation method (to be overridden by subclasses)
    def calculate_salary(self):
        pass

# Subclass FullTimeEmployee
class FullTimeEmployee(Employee):
    def __init__(self, name, employeeID, annual_salary):
        super().__init__(name, employeeID, annual_salary)

    # Override salary calculation for full-time employees
    def calculate_salary(self):
        return self.get_salary()

# Subclass PartTimeEmployee
class PartTimeEmployee(Employee):
    def __init__(self, name, employeeID, hourly_rate, hours_worked):
        super().__init__(name, employeeID, None) # Initialize salary to None
        self.__hourly_rate = hourly_rate
        self.__hours_worked = hours_worked

    def get_hourly_rate(self):
        return self.__hourly_rate

    def set_hourly_rate(self, hourly_rate):
        self.__hourly_rate = hourly_rate

    def get_hours_worked(self):
        return self.__hours_worked

    def set_hours_worked(self, hours_worked):
        self.__hours_worked = hours_worked

    # Override salary calculation for part-time employees
    def calculate_salary(self):
        return self.get_hourly_rate() * self.get_hours_worked()

# Demonstration of polymorphism
full_time_employee = FullTimeEmployee("Shaik Tauheer", "786", 29535)
part_time_employee = PartTimeEmployee("Shaik Mastan Vali", "111", 25, 80)
```

```
print(f"{full_time_employee.get_name()} ({full_time_employee.get_employeeID()})
print(f"Annual Salary: Rs{full_time_employee.calculate_salary()}")

print(f"{part_time_employee.get_name()} ({part_time_employee.get_employeeID()})
print(f"Hourly Rate: Rs {part_time_employee.get_hourly_rate()}")
print(f"Hours Worked: {part_time_employee.get_hours_worked()}")
print(f"Monthly Salary: Rs {part_time_employee.calculate_salary()}")
```

```
Shaik Tauheer (786)
Annual Salary: Rs29535
Shaik Mastan Vali (111)
Hourly Rate: Rs 25
Hours Worked: 80
Monthly Salary: Rs 2000
```

5.Library Management System-Scenario: You are developing a library management system where you need to handle books, patrons, and library transactions.

1. Create a class hierarchy that includes classes for books (e.g., Book), patrons (e.g.. Patron), and transactions (e.g., Transaction). Define attributes and methods for each class.

2. Implement encapsulation by making relevant attributes private and providing getter and setter methods where necessary.

3. Use inheritance to represent different types of books (e.g., fiction, non-fiction) as subclasses of the Book class. Ensure that each book type can have specific attributes and methods.

4. Demonstrate polymorphism by allowing patrons to check out and return books, regardless of the book type.

5. Implement a method for tracking overdue books and notifying patrons.

6. Consider scenarios like book reservations, late fees, and library staff interactions in your design.

In [26]:
```
from datetime import datetime, timedelta

class Book:
    def __init__(self, book_id, title, author):
        self.__book_id = book_id
        self.__title = title
        self.__author = author
        self.__checked_out = False

    def get_book_id(self):
        return self.__book_id

    def get_title(self):
        return self.__title

    def get_author(self):
        return self.__author

    def is_checked_out(self):
        return self.__checked_out

    def check_out(self):
        if not self.__checked_out:
            self.__checked_out = True

    def return_book(self):
        if self.__checked_out:
            self.__checked_out = False

class FictionBook(Book):
    def __init__(self, book_id, title, author, genre):
        super().__init__(book_id, title, author)
        self.__genre = genre

    def get_genre(self):
        return self.__genre

class NonFictionBook(Book):
    def __init__(self, book_id, title, author, subject):
        super().__init__(book_id, title, author)
        self.__subject = subject

    def get_subject(self):
        return self.__subject

class Patron:
    def __init__(self, patron_id, name):
        self.__patron_id = patron_id
        self.__name = name
        self.__checked_out_books = []

    def get_patron_id(self):
        return self.__patron_id

    def get_name(self):
        return self.__name
```

```
    def check_out_book(self, book):
        if not book.is_checked_out():
            book.check_out()
            self.__checked_out_books.append(book)

    def return_book(self, book):
        if book in self.__checked_out_books:
            book.return_book()
            self.__checked_out_books.remove(book)



class Library:
    def __init__(self):
        self.__transactions = []

    def track_transaction(self, transaction):
        self.__transactions.append(transaction)

    def get_overdue_books(self):
        overdue_books = []
        current_date = datetime.now()
        for transaction in self.__transactions:
            if transaction.is_overdue():
                overdue_books.append(transaction.get_book())
        return overdue_books

    def notify_patron(self, patron, message):
        # Implement notification mechanism (e.g., send an email)
        pass


class Transaction:
    def __init__(self, transaction_id, patron, book, due_date):
        self.__transaction_id = transaction_id
        self.__patron = patron
        self.__book = book
        self.__due_date = due_date

    def get_transaction_id(self):
        return self.__transaction_id

    def get_patron(self):
        return self.__patron

    def get_book(self):
        return self.__book

    def is_overdue(self):
        current_date = datetime.now()
        return current_date > self.__due_date

class LibraryStaff:
    @staticmethod
    def reserve_book(book, patron):
        # Implement book reservation logic
        pass
```

```
    @staticmethod
    def calculate_late_fees(patron):
        # Implement late fee calculation logic
        pass

    @staticmethod
    def process_return(transaction):
        # Implement return processing logic
        pass

if __name__ == "__main__":
    book1 = FictionBook(1, "rich dad poor dad", "romio julit", "beast")
    book2 = NonFictionBook(2, "love today", "Half girl firend", "literature of
    patron1 = Patron(786, "Tauheer")
    patron2 = Patron(111, "Mastan Vali")

    library = Library()

    due_date = datetime.now() + timedelta(days=14) # 14 days from today
    transaction1 = Transaction(1, patron1, book1, due_date)
    transaction2 = Transaction(2, patron2, book2, due_date)

    library.track_transaction(transaction1)
    library.track_transaction(transaction2)

    overdue_books = library.get_overdue_books()
    if overdue_books:
        for book in overdue_books:
            print(f"Book '{book.get_title()}' is overdue!")

    patron1.check_out_book(book2)
    print(f"{patron1.get_name()} checked out '{book2.get_title()}'")

    library_staff = LibraryStaff()
    library_staff.process_return(transaction1)
```

```
Tauheer checked out 'love today'
```

6.Online Shopping Cart:

Scenario: You are tasked with designing a class hierarchy for an onlin
e shopping cart system. The system should handle products, shopping ca
rts, and orders. Consider various OOP principles while designing this
system.
1. Create a class hierarchy that includes classes for products (e.g.,
Product), shopping carts (e.g., ShoppingCart), and orders (e.g., Orde

In [25]:
```
class Product:
    def __init__(self, product_id, name, price):
        self.__product_id = product_id # Private attribute for product ID
        self.__name = name # Private attribute for product name
        self.__price = price # Private attribute for product price

    def get_product_id(self):
        return self.__product_id

    def set_product_id(self, product_id):
        self.__product_id = product_id

    def get_name(self):
        return self.__name

    def set_name(self, name):
        self.__name = name

    def get_price(self):
        return self.__price

    def set_price(self, price):
        self.__price = price

    def calculate_discount(self, discount_percentage):
        return self.__price * (discount_percentage / 100)

class ShoppingCart:
    def __init__(self):
        self.__items = [] # Private attribute to store items in the cart

    def add_item(self, product, quantity=1):
        self.__items.append({"product": product, "quantity": quantity})

    def remove_item(self, product):
        self.__items = [item for item in self.__items if item["product"] != pro

    def calculate_total_cost(self):
        total_cost = 0
        for item in self.__items:
            total_cost += item["product"].get_price() * item["quantity"]
        return total_cost

class Order:
    def __init__(self, order_id, items, shipping_address):
        self.__order_id = order_id # Private attribute for order ID
        self.__items = items # Private attribute to store items in the order
        self.__shipping_address = shipping_address # Private attribute for ship

    def get_order_id(self):
        return self.__order_id

    def get_items(self):
        return self.__items

    def get_shipping_address(self):
        return self.__shipping_address
```

```
    def calculate_order_total(self):
        total_cost = 0
        for item in self.__items:
            total_cost += item["product"].get_price() * item["quantity"]
        return total_cost

# Subclasses representing different product types
class Electronics(Product):
    def __init__(self, product_id, name, price, warranty_period):
        super().__init__(product_id, name, price)
        self.__warranty_period = warranty_period

    def get_warranty_period(self):
        return self.__warranty_period

    def set_warranty_period(self, warranty_period):
        self.__warranty_period = warranty_period

class Clothing(Product):
    def __init__(self, product_id, name, price, size, color):
        super().__init__(product_id, name, price)
        self.__size = size
        self.__color = color

    def get_size(self):
        return self.__size

    def set_size(self, size):
        self.__size = size

    def get_color(self):
        return self.__color

    def set_color(self, color):
        self.__color = color
```

In [ ]: