Case Study: Online Shop

An online shop wants to create a software system to manage its products and orders. The system should be designed using object-oriented programming principles, with a focus on inheritance, encapsulation, and polymorphism.

Objectives:

Develop a software system using OOP principles to manage products and orders in an online shop.

Create classes for products, orders, customers, and payment methods with appropriate attributes and behaviors.

Implement inheritance to create subclasses of products and customers.

Implement encapsulation to protect sensitive data and restrict access to certain methods and attributes.

Implement polymorphism to enable flexible and dynamic behavior for methods and classes.

Solution:

To implement the online shop management system, we will use OOP principles and create classes for products, orders, customers, and payment methods.

class Product:

def \_\_init\_\_(self, product\_id, name, description, price, stock\_quantity):

self.product\_id = product\_id

self.name = name

self.description = description

self.price = price

self.\_\_stock\_quantity = stock\_quantity

def add\_stock(self, quantity):

self.\_\_stock\_quantity += quantity

def display(self):

print(f"{self.product\_id} - {self.name}: {self.description} - Price: {self.price}")

class Book(Product):

def \_\_init\_\_(self, product\_id, name, description, price, stock\_quantity, author, publisher):

super().\_\_init\_\_(product\_id, name, description, price, stock\_quantity)

self.author = author

self.publisher = publisher

def display(self):

super().display()

print(f"Author: {self.author} - Publisher: {self.publisher}")

class Customer:

def \_\_init\_\_(self, customer\_id, name, address, email, phone\_number):

self.customer\_id = customer\_id

self.name = name

self.address = address

self.email = email

self.phone\_number = phone\_number

class RegularCustomer(Customer):

pass

class PremiumCustomer(Customer):

def \_\_init\_\_(self, customer\_id, name, address, email, phone\_number, loyalty\_points, discount):

super().\_\_init\_\_(customer\_id, name, address, email, phone\_number)

self.loyalty\_points = loyalty\_points

self.discount = discount

class PaymentMethod:

def \_\_init\_\_(self, payment\_id, name, payment\_type):

self.payment\_id = payment\_id

self.name = name

self.payment\_type = payment\_type

def process\_payment(self, amount):

print(f"Processing {self.payment\_type} payment of {amount} using {self.name}...")

class CreditCard(PaymentMethod):

def \_\_init\_\_(self, payment\_id, name, card\_number, expiration\_date):

super().\_\_init\_\_(payment\_id, name, "credit card")

self.card\_number = card\_number

self.expiration\_date = expiration\_date

class Order:

def \_\_init\_\_(self, order\_id, customer, payment\_method):

self.order\_id = order\_id

self.customer = customer

self.payment\_method = payment\_method

self.items = []

def add\_item(self, product, quantity):

self.items.append((product, quantity))

def remove\_item(self, product):

self.items = [(p, q) for p, q in self.items if p != product]

def calculate\_total\_cost(self):

total = 0

for product, quantity in self.items:

total += product.price \* quantity

return total

def process\_payment(self):

self.payment\_method.process\_payment(self.calculate\_total\_cost())

def display(self):

print(f"Order #{self.order\_id} - Customer: {self.customer.name} - Payment Method: {self.payment\_method.name}")

for product, quantity in self.items:

product.display()

print(f"Quantity: {quantity}")

print(f"Total Cost: {self.calculate\_total\_cost()}")

# Create some products

book1 = Book(1, "The python", "the bestsselling book, 2999, 100, "J.R.R. Tolkien", "Allen & Unwin")

book2 = Book(2, " Philosopher's Stone", "A bestselling children's book", 1499, 50, "J.K. Rowling", "Bloomsbury Publishing")