# 1-misol

# import math

# class Shape:

# def area(self):

# pass

# def perimeter(self):

# pass

# class Circle(Shape):

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

# self.radius = radius

# def area(self):

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

# def perimeter(self):

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

# class Rectangle(Shape):

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

# self.width = width

# self.height = height

# def area(self):

# return self.width \* self.height

# def perimeter(self):

# return 2 \* (self.width + self.height)

# class Triangle(Shape):

# def \_\_init\_\_(self, side1, side2, side3):

# self.side1 = side1

# self.side2 = side2

# self.side3 = side3

# def area(self):

# s = (self.side1 + self.side2 + self.side3) / 2

# return math.sqrt(s \* (s - self.side1) \* (s - self.side2) \* (s - self.side3))

# def perimeter(self):

# return self.side1 + self.side2 + self.side3

# circle = Circle(5)

# print("Circle Area:", circle.area())

# print("Circle Perimeter:", circle.perimeter())

# rectangle = Rectangle(4, 6)

# print("Rectangle Area:", rectangle.area())

# print("Rectangle Perimeter:", rectangle.perimeter())

# triangle = Triangle(3, 4, 5)

# print("Triangle Area:", triangle.area())

# print("Triangle Perimeter:", triangle.perimeter())

# 2-misol

# class Account:

# def \_\_init\_\_(self, balance=0):

# self.\_balance = balance

# def deposit(self, amount):

# self.\_balance += amount

# def withdraw(self, amount):

# if self.\_balance >= amount:

# self.\_balance -= amount

# else:

# print("Insufficient funds")

# def calculate\_interest(self):

# pass

# class SavingsAccount(Account):

# def \_\_init\_\_(self, balance=0, interest\_rate=0.01):

# super().\_\_init\_\_(balance)

# self.\_interest\_rate = interest\_rate

# def calculate\_interest(self):

# return self.\_balance \* self.\_interest\_rate

# class CheckingAccount(Account):

# def \_\_init\_\_(self, balance=0, transaction\_fee=1.0):

# super().\_\_init\_\_(balance)

# self.\_transaction\_fee = transaction\_fee

# def withdraw(self, amount):

# if self.\_balance >= amount + self.\_transaction\_fee:

# self.\_balance -= amount + self.\_transaction\_fee

# else:

# print("Insufficient funds")

# def calculate\_interest(self):

# return 0

# def calculate\_total\_interest(accounts):

# total\_interest = 0

# for account in accounts:

# total\_interest += account.calculate\_interest()

# return total\_interest

# savings\_account = SavingsAccount(1000, 0.02)

# checking\_account = CheckingAccount(2000, 0.5)

# accounts = [savings\_account, checking\_account]

# total\_interest\_earned = calculate\_total\_interest(accounts)

# print("Total interest earned: $", total\_interest\_earned)

# 3-misol

# class Employee:

# def \_\_init\_\_(self, name, salary, performance):

# self.name = name

# self.salary = salary

# self.performance = performance

# def calculate\_bonus(self):

# raise NotImplementedError("Subclasses must implement calculate\_bonus method")

# class Manager(Employee):

# def calculate\_bonus(self):

# bonus = self.salary \* self.performance \* 0.1

# return bonus

# class Engineer(Employee):

# def calculate\_bonus(self):

# bonus = self.salary \* self.performance \* 0.05

# return bonus

# class Salesperson(Employee):

# def calculate\_bonus(self):

# bonus = self.salary \* self.performance \* 0.08

# return bonus

# employees = [

# Manager("Johogir", 50000, 0.9),

# Engineer("Alisher", 40000, 0.8),

# Salesperson("Bobur", 30000, 0.7)

# ]

# for employee in employees:

# bonus = employee.calculate\_bonus()

# print(f"{employee.name}: Bonus = ${bonus:.2f}")

# 4-misol

# class LibraryItem:

# def \_\_init\_\_(self, title, author):

# self.\_title = title

# self.\_author = author

# def display\_details(self):

# pass

# def get\_title(self):

# return self.\_title

# def get\_author(self):

# return self.\_author

# class Book(LibraryItem):

# def \_\_init\_\_(self, title, author, isbn):

# super().\_\_init\_\_(title, author)

# self.\_isbn = isbn

# def display\_details(self):

# print("Book Details:")

# print("Title:", self.get\_title())

# print("Author:", self.get\_author())

# print("ISBN:", self.\_isbn)

# class DVD(LibraryItem):

# def \_\_init\_\_(self, title, author, director):

# super().\_\_init\_\_(title, author)

# self.\_director = director

# def display\_details(self):

# print("DVD Details:")

# print("Title:", self.get\_title())

# print("Author:", self.get\_author())

# print("Director:", self.\_director)

# book1 = Book("Python Programming", "John Smith", "978-3-16-148410-0")

# dvd1 = DVD("The Matrix", "Lana Wachowski", "Lana Wachowski")

# book1.display\_details()

# print()

# dvd1.display\_details()

# 5-misol

# class Animal:

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

# self.name = name

# def sound(self):

# pass

# def movement(self):

# pass

# class Mammal(Animal):

# def sound(self):

# return "Mammal sound"

# def movement(self):

# return "Mammal movement"

# class Bird(Animal):

# def sound(self):

# return "Bird sound"

# def movement(self):

# return "Bird movement"

# class Fish(Animal):

# def sound(self):

# return "Fish sound"

# def movement(self):

# return "Fish movement"

# def simulate\_animal(animal):

# print(f"{animal.name}:")

# print("Sound:", animal.sound())

# print("Movement:", animal.movement())

# print()

# lion = Mammal("Lion")

# parrot = Bird("Parrot")

# shark = Fish("Shark")

# simulate\_animal(lion)

# simulate\_animal(parrot)

# simulate\_animal(shark)

# 6-misol

# import random

# class Vehicle:

# def \_\_init\_\_(self, speed, passenger\_capacity):

# self.\_speed = speed

# self.\_passenger\_capacity = passenger\_capacity

# def accelerate(self, acceleration):

# self.\_speed += acceleration

# def brake(self, deceleration):

# if self.\_speed - deceleration >= 0:

# self.\_speed -= deceleration

# else:

# self.\_speed = 0

# def get\_speed(self):

# return self.\_speed

# def get\_passenger\_capacity(self):

# return self.\_passenger\_capacity

# def react\_to\_traffic(self):

# print("Vehicle is reacting to traffic.")

# class Car(Vehicle):

# def react\_to\_traffic(self):

# print("Car is changing lanes.")

# class Bicycle(Vehicle):

# def react\_to\_traffic(self):

# print("Bicycle is slowing down.")

# class Truck(Vehicle):

# def react\_to\_traffic(self):

# print("Truck is maintaining a safe distance.")

# def simulate\_traffic():

# vehicles = []

# vehicle\_types = [Car, Bicycle, Truck]

# for \_ in range(10):

# vehicle\_type = random.choice(vehicle\_types)

# speed = random.randint(0, 100)

# passenger\_capacity = random.randint(1, 5)

# vehicle = vehicle\_type(speed, passenger\_capacity)

# vehicles.append(vehicle)

# for vehicle in vehicles:

# vehicle.react\_to\_traffic()

# simulate\_traffic()

# 7-misol

# class University:

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

# self.name = name

# self.departments = []

# def add\_department(self, department):

# self.departments.append(department)

# def get\_departments(self):

# return self.departments

# class Department:

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

# self.name = name

# self.professors = []

# self.students = []

# def add\_professor(self, professor):

# self.professors.append(professor)

# def add\_student(self, student):

# self.students.append(student)

# def get\_professors(self):

# return self.professors

# def get\_students(self):

# return self.students

# class Person:

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

# self.name = name

# class Professor(Person):

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

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

# self.courses\_taught = []

# def add\_course(self, course):

# self.courses\_taught.append(course)

# def assign\_grade(self, student, course, grade):

# student.add\_grade(course, grade)

# class Student(Person):

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

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

# self.courses\_taken = {}

# self.research\_projects = []

# def add\_course(self, course):

# self.courses\_taken[course] = None

# def add\_grade(self, course, grade):

# self.courses\_taken[course] = grade

# def add\_research\_project(self, project):

# self.research\_projects.append(project)

# def assign\_research\_projects(person, project):

# person.add\_research\_project(project)

# university = University("My University")

# department1 = Department("Computer Science")

# department2 = Department("Physics")

# professor1 = Professor("John Smith")

# professor2 = Professor("Jane Doe")

# student1 = Student("Alice Johnson")

# student2 = Student("Bob Williams")

# department1.add\_professor(professor1)

# department1.add\_student(student1)

# department2.add\_professor(professor2)

# department2.add\_student(student2)

# university.add\_department(department1)

# university.add\_department(department2)

# research\_project = "Machine Learning"

# assign\_research\_projects(professor1, research\_project)

# assign\_research\_projects(student2, research\_project)

# professor1.assign\_grade(student1, "Python Programming", "A")

# 8-misol

# class User:

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

# self.name = name

# self.email = email

# def login(self):

# print("Logged in as", self.name)

# class Customer(User):

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

# super().\_\_init\_\_(name, email)

# self.cart = ShoppingCart()

# def purchase(self):

# self.cart.checkout(self)

# class Seller(User):

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

# super().\_\_init\_\_(name, email)

# self.products = []

# def add\_product(self, product):

# self.products.append(product)

# def remove\_product(self, product):

# self.products.remove(product)

# class Product:

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

# self.name = name

# self.price = price

# class ShoppingCart:

# def \_\_init\_\_(self):

# self.items = []

# def add\_item(self, product):

# self.items.append(product)

# def remove\_item(self, product):

# self.items.remove(product)

# def checkout(self, user):

# total = 0

# print("Order Summary:")

# for item in self.items:

# print(item.name, "-", item.price)

# total += item.price

# print("Total:", total)

# payment\_method = input("Select payment method: ")

# shipping\_option = input("Select shipping option: ")

# order = Order(self.items, total, payment\_method, shipping\_option, user)

# order.process()

# class Order:

# def \_\_init\_\_(self, items, total, payment\_method, shipping\_option, user):

# self.items = items

# self.total = total

# self.payment\_method = payment\_method

# self.shipping\_option = shipping\_option

# self.user = user

# def process(self):

# print("Processing order...")

# print("Payment method:", self.payment\_method)

# print("Shipping option:", self.shipping\_option)

# print("Order processed successfully!")

# customer = Customer("John Doe", "john@example.com")

# customer.login()

# seller = Seller("Jane Smith", "jane@example.com")

# seller.login()

# product1 = Product("Product 1", 10)

# product2 = Product("Product 2", 20)

# seller.add\_product(product1)

# seller.add\_product(product2)

# customer.cart.add\_item(product1)

# customer.cart.add\_item(product2)

# customer.cart.checkout(customer)

# 9-misol

# class User:

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

# self.name = name

# self.email = email

# def display\_info(self):

# print(f"Name: {self.name}")

# print(f"Email: {self.email}")

# class Customer(User):

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

# super().\_\_init\_\_(name, email)

# self.shopping\_cart = ShoppingCart()

# def purchase\_items(self):

# order = Order(self.shopping\_cart)

# order.process()

# class Seller(User):

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

# super().\_\_init\_\_(name, email)

# def add\_product(self, product):

# pass

# class Product:

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

# self.name = name

# self.price = price

# class ShoppingCart:

# def \_\_init\_\_(self):

# self.items = []

# def add\_item(self, product):

# self.items.append(product)

# def remove\_item(self, product):

# self.items.remove(product)

# def calculate\_total(self):

# total = 0

# for item in self.items:

# total += item.price

# return total

# class Order:

# def \_\_init\_\_(self, cart):

# self.cart = cart

# def process(self):

# payment\_method = self.select\_payment\_method()

# shipping\_option = self.select\_shipping\_option()

# self.cart.calculate\_total()

# def select\_payment\_method(self):

# pass

# def select\_shipping\_option(self):

# pass

# customer1 = Customer("John", "john@example.com")

# customer1.display\_info()

# seller1 = Seller("Jane", "jane@example.com")

# seller1.display\_info()

# product1 = Product("Product 1", 10.99)

# product2 = Product("Product 2", 19.99)

# seller1.add\_product(product1)

# seller1.add\_product(product2)

# customer1.shopping\_cart.add\_item(product1)

# customer1.shopping\_cart.add\_item(product2)

# customer1.shopping\_cart.remove\_item(product1)

# customer1.purchase\_items()

# 11-misol

# class Character:

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

# self.name = name

# self.health = health

# def attack(self, target):

# pass

# def take\_damage(self, damage):

# self.health -= damage

# if self.health <= 0:

# print(f"{self.name} has been defeated!")

# class PlayerCharacter(Character):

# def \_\_init\_\_(self, name, health, level):

# super().\_\_init\_\_(name, health)

# self.level = level

# def attack(self, target):

# print(f"{self.name} attacks {target.name}!")

# target.take\_damage(self.level \* 10)

# class NonPlayerCharacter(Character):

# def \_\_init\_\_(self, name, health, quest):

# super().\_\_init\_\_(name, health)

# self.quest = quest

# def attack(self, target): # 1-misol

# import math

# class Shape:

# def area(self):

# pass

# def perimeter(self):

# pass

# class Circle(Shape):

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

# self.radius = radius

# def area(self):

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

# def perimeter(self):

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

# class Rectangle(Shape):

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

# self.width = width

# self.height = height

# def area(self):

# return self.width \* self.height

# def perimeter(self):

# return 2 \* (self.width + self.height)

# class Triangle(Shape):

# def \_\_init\_\_(self, side1, side2, side3):

# self.side1 = side1

# self.side2 = side2

# self.side3 = side3

# def area(self):

# s = (self.side1 + self.side2 + self.side3) / 2

# return math.sqrt(s \* (s - self.side1) \* (s - self.side2) \* (s - self.side3))

# def perimeter(self):

# return self.side1 + self.side2 + self.side3

# circle = Circle(5)

# print("Circle Area:", circle.area())

# print("Circle Perimeter:", circle.perimeter())

# rectangle = Rectangle(4, 6)

# print("Rectangle Area:", rectangle.area())

# print("Rectangle Perimeter:", rectangle.perimeter())

# triangle = Triangle(3, 4, 5)

# print("Triangle Area:", triangle.area())

# print("Triangle Perimeter:", triangle.perimeter())

# 2-misol

# class Account:

# def \_\_init\_\_(self, balance=0):

# self.\_balance = balance

# def deposit(self, amount):

# self.\_balance += amount

# def withdraw(self, amount):

# if self.\_balance >= amount:

# self.\_balance -= amount

# else:

# print("Insufficient funds")

# def calculate\_interest(self):

# pass

# class SavingsAccount(Account):

# def \_\_init\_\_(self, balance=0, interest\_rate=0.01):

# super().\_\_init\_\_(balance)

# self.\_interest\_rate = interest\_rate

# def calculate\_interest(self):

# return self.\_balance \* self.\_interest\_rate

# class CheckingAccount(Account):

# def \_\_init\_\_(self, balance=0, transaction\_fee=1.0):

# super().\_\_init\_\_(balance)

# self.\_transaction\_fee = transaction\_fee

# def withdraw(self, amount):

# if self.\_balance >= amount + self.\_transaction\_fee:

# self.\_balance -= amount + self.\_transaction\_fee

# else:

# print("Insufficient funds")

# def calculate\_interest(self):

# return 0

# def calculate\_total\_interest(accounts):

# total\_interest = 0

# for account in accounts:

# total\_interest += account.calculate\_interest()

# return total\_interest

# savings\_account = SavingsAccount(1000, 0.02)

# checking\_account = CheckingAccount(2000, 0.5)

# accounts = [savings\_account, checking\_account]

# total\_interest\_earned = calculate\_total\_interest(accounts)

# print("Total interest earned: $", total\_interest\_earned)

# 3-misol

# class Employee:

# def \_\_init\_\_(self, name, salary, performance):

# self.name = name

# self.salary = salary

# self.performance = performance

# def calculate\_bonus(self):

# raise NotImplementedError("Subclasses must implement calculate\_bonus method")

# class Manager(Employee):

# def calculate\_bonus(self):

# bonus = self.salary \* self.performance \* 0.1

# return bonus

# class Engineer(Employee):

# def calculate\_bonus(self):

# bonus = self.salary \* self.performance \* 0.05

# return bonus

# class Salesperson(Employee):

# def calculate\_bonus(self):

# bonus = self.salary \* self.performance \* 0.08

# return bonus

# employees = [

# Manager("Johogir", 50000, 0.9),

# Engineer("Alisher", 40000, 0.8),

# Salesperson("Bobur", 30000, 0.7)

# ]

# for employee in employees:

# bonus = employee.calculate\_bonus()

# print(f"{employee.name}: Bonus = ${bonus:.2f}")

# 4-misol

# class LibraryItem:

# def \_\_init\_\_(self, title, author):

# self.\_title = title

# self.\_author = author

# def display\_details(self):

# pass

# def get\_title(self):

# return self.\_title

# def get\_author(self):

# return self.\_author

# class Book(LibraryItem):

# def \_\_init\_\_(self, title, author, isbn):

# super().\_\_init\_\_(title, author)

# self.\_isbn = isbn

# def display\_details(self):

# print("Book Details:")

# print("Title:", self.get\_title())

# print("Author:", self.get\_author())

# print("ISBN:", self.\_isbn)

# class DVD(LibraryItem):

# def \_\_init\_\_(self, title, author, director):

# super().\_\_init\_\_(title, author)

# self.\_director = director

# def display\_details(self):

# print("DVD Details:")

# print("Title:", self.get\_title())

# print("Author:", self.get\_author())

# print("Director:", self.\_director)

# book1 = Book("Python Programming", "John Smith", "978-3-16-148410-0")

# dvd1 = DVD("The Matrix", "Lana Wachowski", "Lana Wachowski")

# book1.display\_details()

# print()

# dvd1.display\_details()

# 5-misol

# class Animal:

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

# self.name = name

# def sound(self):

# pass

# def movement(self):

# pass

# class Mammal(Animal):

# def sound(self):

# return "Mammal sound"

# def movement(self):

# return "Mammal movement"

# class Bird(Animal):

# def sound(self):

# return "Bird sound"

# def movement(self):

# return "Bird movement"

# class Fish(Animal):

# def sound(self):

# return "Fish sound"

# def movement(self):

# return "Fish movement"

# def simulate\_animal(animal):

# print(f"{animal.name}:")

# print("Sound:", animal.sound())

# print("Movement:", animal.movement())

# print()

# lion = Mammal("Lion")

# parrot = Bird("Parrot")

# shark = Fish("Shark")

# simulate\_animal(lion)

# simulate\_animal(parrot)

# simulate\_animal(shark)

# 6-misol

# import random

# class Vehicle:

# def \_\_init\_\_(self, speed, passenger\_capacity):

# self.\_speed = speed

# self.\_passenger\_capacity = passenger\_capacity

# def accelerate(self, acceleration):

# self.\_speed += acceleration

# def brake(self, deceleration):

# if self.\_speed - deceleration >= 0:

# self.\_speed -= deceleration

# else:

# self.\_speed = 0

# def get\_speed(self):

# return self.\_speed

# def get\_passenger\_capacity(self):

# return self.\_passenger\_capacity

# def react\_to\_traffic(self):

# print("Vehicle is reacting to traffic.")

# class Car(Vehicle):

# def react\_to\_traffic(self):

# print("Car is changing lanes.")

# class Bicycle(Vehicle):

# def react\_to\_traffic(self):

# print("Bicycle is slowing down.")

# class Truck(Vehicle):

# def react\_to\_traffic(self):

# print("Truck is maintaining a safe distance.")

# def simulate\_traffic():

# vehicles = []

# vehicle\_types = [Car, Bicycle, Truck]

# for \_ in range(10):

# vehicle\_type = random.choice(vehicle\_types)

# speed = random.randint(0, 100)

# passenger\_capacity = random.randint(1, 5)

# vehicle = vehicle\_type(speed, passenger\_capacity)

# vehicles.append(vehicle)

# for vehicle in vehicles:

# vehicle.react\_to\_traffic()

# simulate\_traffic()

# 7-misol

# class University:

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

# self.name = name

# self.departments = []

# def add\_department(self, department):

# self.departments.append(department)

# def get\_departments(self):

# return self.departments

# class Department:

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

# self.name = name

# self.professors = []

# self.students = []

# def add\_professor(self, professor):

# self.professors.append(professor)

# def add\_student(self, student):

# self.students.append(student)

# def get\_professors(self):

# return self.professors

# def get\_students(self):

# return self.students

# class Person:

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

# self.name = name

# class Professor(Person):

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

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

# self.courses\_taught = []

# def add\_course(self, course):

# self.courses\_taught.append(course)

# def assign\_grade(self, student, course, grade):

# student.add\_grade(course, grade)

# class Student(Person):

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

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

# self.courses\_taken = {}

# self.research\_projects = []

# def add\_course(self, course):

# self.courses\_taken[course] = None

# def add\_grade(self, course, grade):

# self.courses\_taken[course] = grade

# def add\_research\_project(self, project):

# self.research\_projects.append(project)

# def assign\_research\_projects(person, project):

# person.add\_research\_project(project)

# university = University("My University")

# department1 = Department("Computer Science")

# department2 = Department("Physics")

# professor1 = Professor("John Smith")

# professor2 = Professor("Jane Doe")

# student1 = Student("Alice Johnson")

# student2 = Student("Bob Williams")

# department1.add\_professor(professor1)

# department1.add\_student(student1)

# department2.add\_professor(professor2)

# department2.add\_student(student2)

# university.add\_department(department1)

# university.add\_department(department2)

# research\_project = "Machine Learning"

# assign\_research\_projects(professor1, research\_project)

# assign\_research\_projects(student2, research\_project)

# professor1.assign\_grade(student1, "Python Programming", "A")

# 8-misol

# class User:

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

# self.name = name

# self.email = email

# def login(self):

# print("Logged in as", self.name)

# class Customer(User):

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

# super().\_\_init\_\_(name, email)

# self.cart = ShoppingCart()

# def purchase(self):

# self.cart.checkout(self)

# class Seller(User):

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

# super().\_\_init\_\_(name, email)

# self.products = []

# def add\_product(self, product):

# self.products.append(product)

# def remove\_product(self, product):

# self.products.remove(product)

# class Product:

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

# self.name = name

# self.price = price

# class ShoppingCart:

# def \_\_init\_\_(self):

# self.items = []

# def add\_item(self, product):

# self.items.append(product)

# def remove\_item(self, product):

# self.items.remove(product)

# def checkout(self, user):

# total = 0

# print("Order Summary:")

# for item in self.items:

# print(item.name, "-", item.price)

# total += item.price

# print("Total:", total)

# payment\_method = input("Select payment method: ")

# shipping\_option = input("Select shipping option: ")

# order = Order(self.items, total, payment\_method, shipping\_option, user)

# order.process()

# class Order:

# def \_\_init\_\_(self, items, total, payment\_method, shipping\_option, user):

# self.items = items

# self.total = total

# self.payment\_method = payment\_method

# self.shipping\_option = shipping\_option

# self.user = user

# def process(self):

# print("Processing order...")

# print("Payment method:", self.payment\_method)

# print("Shipping option:", self.shipping\_option)

# print("Order processed successfully!")

# customer = Customer("John Doe", "john@example.com")

# customer.login()

# seller = Seller("Jane Smith", "jane@example.com")

# seller.login()

# product1 = Product("Product 1", 10)

# product2 = Product("Product 2", 20)

# seller.add\_product(product1)

# seller.add\_product(product2)

# customer.cart.add\_item(product1)

# customer.cart.add\_item(product2)

# customer.cart.checkout(customer)

# 9-misol

# class User:

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

# self.name = name

# self.email = email

# def display\_info(self):

# print(f"Name: {self.name}")

# print(f"Email: {self.email}")

# class Customer(User):

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

# super().\_\_init\_\_(name, email)

# self.shopping\_cart = ShoppingCart()

# def purchase\_items(self):

# order = Order(self.shopping\_cart)

# order.process()

# class Seller(User):

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

# super().\_\_init\_\_(name, email)

# def add\_product(self, product):

# pass

# class Product:

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

# self.name = name

# self.price = price

# class ShoppingCart:

# def \_\_init\_\_(self):

# self.items = []

# def add\_item(self, product):

# self.items.append(product)

# def remove\_item(self, product):

# self.items.remove(product)

# def calculate\_total(self):

# total = 0

# for item in self.items:

# total += item.price

# return total

# class Order:

# def \_\_init\_\_(self, cart):

# self.cart = cart

# def process(self):

# payment\_method = self.select\_payment\_method()

# shipping\_option = self.select\_shipping\_option()

# self.cart.calculate\_total()

# def select\_payment\_method(self):

# pass

# def select\_shipping\_option(self):

# pass

# customer1 = Customer("John", "john@example.com")

# customer1.display\_info()

# seller1 = Seller("Jane", "jane@example.com")

# seller1.display\_info()

# product1 = Product("Product 1", 10.99)

# product2 = Product("Product 2", 19.99)

# seller1.add\_product(product1)

# seller1.add\_product(product2)

# customer1.shopping\_cart.add\_item(product1)

# customer1.shopping\_cart.add\_item(product2)

# customer1.shopping\_cart.remove\_item(product1)

# customer1.purchase\_items()

# 11-misol

# class Character:

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

# self.name = name

# self.health = health

# def attack(self, target):

# pass

# def take\_damage(self, damage):

# self.health -= damage

# if self.health <= 0:

# print(f"{self.name} has been defeated!")

# class PlayerCharacter(Character):

# def \_\_init\_\_(self, name, health, level):

# super().\_\_init\_\_(name, health)

# self.level = level

# def attack(self, target):

# print(f"{self.name} attacks {target.name}!")

# target.take\_damage(self.level \* 10)

# class NonPlayerCharacter(Character):

# def \_\_init\_\_(self, name, health, quest):

# super().\_\_init\_\_(name, health)

# self.quest = quest

# def attack(self, target):

# print(f"{self.name} attacks {target.name}!")

# target.take\_damage(5)

# class Item:

# def \_\_init\_\_(self, name, damage\_bonus):

# self.name = name

# self.damage\_bonus = damage\_bonus

# class Quest:

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

# self.name = name

# self.reward = reward

# class GameWorld:

# def \_\_init\_\_(self):

# self.characters = []

# self.items = []

# self.quests = []

# def add\_character(self, character):

# self.characters.append(character)

# def add\_item(self, item):

# self.items.append(item)

# def add\_quest(self, quest):

# self.quests.append(quest)

# def start\_battle(self, player, enemy):

# player.attack(enemy)

# enemy.attack(player)

# def showcase\_polymorphism(self):

# for character in self.characters:

# character.attack(self.characters[0])

# world = GameWorld()

# quest1 = Quest("Retrieve the Artifact", "Experience Points")

# quest2 = Quest("Defeat the Boss", "Legendary Weapon")

# world.add\_quest(quest1)

# world.add\_quest(quest2)

# item1 = Item("Sword", 5)

# item2 = Item("Bow", 7)

# world.add\_item(item1)

# world.add\_item(item2)

# player = PlayerCharacter("Hero", 100, 5)

# npc1 = NonPlayerCharacter("Villain", 50, quest1)

# npc2 = NonPlayerCharacter("Boss", 200, quest2)

# world.add\_character(player)

# world.add\_character(npc1)

# world.add\_character(npc2)

# world.start\_battle(player, npc1)

# world.showcase\_polymorphism()

# print(f"{self.name} attacks {target.name}!")

# target.take\_damage(5)

# class Item:

# def \_\_init\_\_(self, name, damage\_bonus):

# self.name = name

# self.damage\_bonus = damage\_bonus

# class Quest:

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

# self.name = name

# self.reward = reward

# class GameWorld:

# def \_\_init\_\_(self):

# self.characters = []

# self.items = []

# self.quests = []

# def add\_character(self, character):

# self.characters.append(character)

# def add\_item(self, item):

# self.items.append(item)

# def add\_quest(self, quest):

# self.quests.append(quest)

# def start\_battle(self, player, enemy):

# player.attack(enemy)

# enemy.attack(player)

# def showcase\_polymorphism(self):

# for character in self.characters:

# character.attack(self.characters[0])

# world = GameWorld()

# quest1 = Quest("Retrieve the Artifact", "Experience Points")

# quest2 = Quest("Defeat the Boss", "Legendary Weapon")

# world.add\_quest(quest1)

# world.add\_quest(quest2)

# item1 = Item("Sword", 5)

# item2 = Item("Bow", 7)

# world.add\_item(item1)

# world.add\_item(item2)

# player = PlayerCharacter("Hero", 100, 5)

# npc1 = NonPlayerCharacter("Villain", 50, quest1)

# npc2 = NonPlayerCharacter("Boss", 200, quest2)

# world.add\_character(player)

# world.add\_character(npc1)

# world.add\_character(npc2)

# world.start\_battle(player, npc1)

# world.showcase\_polymorphism()