OOPs Practice Question

PRACTICE QUESTIONS

1. Smart Home Automation System

Question: Design a smart home automation system. Create a base class `Device` with common attributes like `device_id`, `name`, and methods like `turn_on` and `turn_off`. Create derived classes `Light`, `Thermostat`, and `SecurityCamera` with specific attributes and methods. Implement polymorphism to handle different devices.

Requirements:

- `Device` class with `device_id`, `name`, `turn_on()`, and `turn_off()`.
- `Light` class with `brightness` and `color` attributes.
- `Thermostat` class with `temperature` attribute and `set_temperature()` method.
- `SecurityCamera` class with `resolution` attribute and `record()` method.
- Use polymorphism to iterate over a list of devices and call their specific methods.

2. Employee Management System

Question: Develop an employee management system. Create a base class `Employee` with attributes like `name`, `id`, and `salary`. Create derived classes `Manager`, `Developer`, and `Designer` with specific attributes and methods. Implement method overriding and class-specific behavior.

Requirements:

- `Employee` class with `name`, `id`, `salary`, and `display_details()` method.
- `Manager` class with `team_size` attribute and `conduct_meeting()` method.
- `Developer` class with `programming_languages` attribute and `write_code()` method.
- `Designer` class with `design_tools` attribute and `create_design()` method.
- Override `display_details()` in each subclass to include specific details.

3. E-Learning Platform

Question: Create an e-learning platform. Design a base class `Course` with common attributes like `course_id`, `title`, and methods like `enroll_student` and `unenroll_student`. Create derived classes `ProgrammingCourse`, `DesignCourse`, and `MathCourse` with specific attributes and methods.

Requirements:

- `Course` class with `course_id`, `title`, `students`, `enroll_student()`, and `unenroll_student()`.

- `ProgrammingCourse` class with `languages` attribute and `start_project()` method.
- `DesignCourse` class with `software` attribute and `assign_design_task()` method.
- `MathCourse` class with `difficulty_level` attribute and `assign_homework()` method.
- Use polymorphism to manage courses and their specific tasks.

4. Restaurant Management System

Question: Design a restaurant management system. Create a base class `Restaurant` with common attributes like `name`, `location`, and methods like `open_restaurant` and `close_restaurant`. Create derived classes `FastFoodRestaurant`, `FineDiningRestaurant`, and `Cafe` with specific attributes and methods.

Requirements:

- `Restaurant` class with `name`, `location`, `open_restaurant()`, and `close_restaurant()`.
- `FastFoodRestaurant` class with `drive_thru` attribute and `serve_fast_food()` method.
- `FineDiningRestaurant` class with `dress_code` attribute and `serve_gourmet_dishes()` method.
- `Cafe` class with `coffee_types` attribute and `serve_coffee()` method.
- Override `open_restaurant()` and `close_restaurant()` in each subclass.

5. Hospital Management System

Question: Develop a hospital management system. Create a base class `Person` with common attributes like `name`, `age`, and methods like `display_info`. Create derived classes `Doctor`, `Nurse`, and `Patient` with specific attributes and methods. Implement polymorphism and method overriding.

Requirements:

- `Person` class with `name`, `age`, and `display_info()`.
- `Doctor` class with `specialization` attribute and `diagnose()` method.
- `Nurse` class with `shift` attribute and `assist()` method.
- `Patient` class with `ailment` attribute and `receive_treatment()` method.
- Override `display_info()` in each subclass to include specific details.

6. Online Booking System

Question: Create an online booking system. Design a base class `Booking` with common attributes like `booking_id`, `date`, and methods like `confirm_booking` and `cancel_booking`. Create derived classes `HotelBooking`, `FlightBooking`, and `EventBooking` with specific attributes and methods.

Requirements:

- `Booking` class with `booking_id`, `date`, `confirm_booking()`, and `cancel_booking()`.
- `HotelBooking` class with `hotel_name` attribute and `reserve_room()` method.
- `FlightBooking` class with `flight_number` attribute and `book_seat()` method.
- `EventBooking` class with `event_name` attribute and `reserve_ticket()` method.
- Use polymorphism to manage different bookings.

7. University Course Management

Question: Design a university course management system. Create a base class `UniversityMember` with common attributes like `name`, `id`, and methods like `show_details`. Create derived classes `Professor`, `Student`, and `Administrator` with specific attributes and methods.

Requirements:

- `UniversityMember` class with `name`, `id`, and `show_details()`.
- `Professor` class with `department` attribute and `teach_course()` method.
- `Student` class with `major` attribute and `enroll_course()` method.
- `Administrator` class with `role` attribute and `manage_operations()` method.
- Override `show_details()` in each subclass to include specific details.

8. Transport System

Question: Develop a transport system. Create a base class `Transport` with common attributes like `id`, `capacity`, and methods like `start` and `stop`. Create derived classes `Bus`, `Train`, and `Airplane` with specific attributes and methods. Implement method overriding and polymorphism.

Requirements:

- `Transport` class with `id`, `capacity`, `start()`, and `stop()`.
- `Bus` class with `route_number` attribute and `pick_up_passengers()` method.
- `Train` class with `number_of_coaches` attribute and `depart()` method.
- `Airplane` class with `flight_code` attribute and `take_off()` method.
- Use polymorphism to manage different transport types.

9. Smart City Infrastructure

Question: Create a smart city infrastructure management system. Design a base class `Infrastructure` with common attributes like `location`, `status`, and methods like `activate` and `deactivate`. Create derived classes `TrafficLight`, `StreetLight`, and `SurveillanceCamera` with specific attributes and methods.

Requirements:

- `Infrastructure` class with `location`, `status`, `activate()`, and `deactivate()`.
- `TrafficLight` class with `intersection_id` attribute and `change_light()` method.

- `StreetLight` class with `lumens` attribute and `adjust_brightness()` method.
- `SurveillanceCamera` class with `resolution` attribute and `start_recording()` method.
- Override `activate()` and `deactivate()` in each subclass.

10. Game Development System

Question: Design a game development system. Create a base class `GameCharacter` with common attributes like `name`, `health`, and methods like `move` and `attack`. Create derived classes `Warrior`, `Mage`, and `Archer` with specific attributes and methods.

Requirements:

- `GameCharacter` class with `name`, `health`, `move()`, and `attack()`.
- `Warrior` class with `strength` attribute and `use_sword()` method.
- `Mage` class with `mana` attribute and `cast_spell()` method.
- `Archer` class with `range` attribute and `shoot_arrow()` method.
- Override `attack()` in each subclass to provide specific attack behaviors.

11. E-Commerce Recommendation System

Question: Create an e-commerce recommendation system. Design a base class `Product` with common attributes like `product_id`, `name`, and methods like `get_details`. Create derived classes `Electronics`, `Clothing`, and `Books` with specific attributes and methods.

Requirements:

- `Product` class with `product_id`, `name`, and `get_details()`.
- `Electronics` class with `warranty` attribute and `get_warranty_info()` method.
- `Clothing` class with `size` attribute and `get_size_chart()` method.
- `Books` class with `author` attribute and `get_author_info()` method.
- Use polymorphism to display product details based on type.

12. Financial Portfolio Management

Question: Develop a financial portfolio management system. Create a base class `Asset` with common attributes like `asset_id`, `value`, and methods like `calculate_return`. Create derived classes `Stock`, `Bond`, and `RealEstate` with specific attributes and methods.

Requirements:

- `Asset` class with `asset_id`, `value`, and `calculate_return()`.
- `Stock` class with `ticker_symbol` attribute and `calculate_dividend()` method.
- `Bond` class with `interest_rate` attribute and `calculate_interest()` method.
- `RealEstate` class with `location` attribute and `calculate_rent()` method.
- Override `calculate_return()` in each subclass to include specific calculations.

```
class Device:
    def __init__(self, device_id, name):
```

```
        self.device_id = device_id
        self.name = name

    def turn_on(self):
        print(f"{self.name} (ID: {self.device_id}) is now ON.")

    def turn_off(self):
        print(f"{self.name} (ID: {self.device_id}) is now OFF.")


class Light(Device):
    def __init__(self, device_id, name, brightness, color):
        super().__init__(device_id, name)
        self.brightness = brightness
        self.color = color

    def set_brightness(self, brightness):
        self.brightness = brightness
        print(f"{self.name} brightness set to {self.brightness}.")

    def set_color(self, color):
        self.color = color
        print(f"{self.name} color set to {self.color}.")


class Thermostat(Device):
    def __init__(self, device_id, name, temperature):
        super().__init__(device_id, name)
        self.temperature = temperature

    def set_temperature(self, temperature):
        self.temperature = temperature
        print(f"{self.name} temperature set to {self.temperature}
degrees.")


class SecurityCamera(Device):
    def __init__(self, device_id, name, resolution):
        super().__init__(device_id, name)
        self.resolution = resolution

    def record(self):
        print(f"{self.name} is recording at {self.resolution}
resolution.")


# Demonstrating polymorphism
devices = [
    Light(device_id=1, name="Living Room Light", brightness=75,
color="Warm White"),
    Thermostat(device_id=2, name="Bedroom Thermostat",
```

```
temperature=22),
    SecurityCamera(device_id=3, name="Front Door Camera",
resolution="1080p")
]

for device in devices:
    device.turn_on()

devices[0].set_brightness(80)
devices[1].set_temperature(24)
devices[2].record()

for device in devices:
    device.turn_off()
```

```
Living Room Light (ID: 1) is now ON.
Bedroom Thermostat (ID: 2) is now ON.
Front Door Camera (ID: 3) is now ON.
Living Room Light brightness set to 80.
Bedroom Thermostat temperature set to 24 degrees.
Front Door Camera is recording at 1080p resolution.
Living Room Light (ID: 1) is now OFF.
Bedroom Thermostat (ID: 2) is now OFF.
Front Door Camera (ID: 3) is now OFF.
```

```
class Employee:
    def __init__(self, name, id, salary):
        self.name = name
        self.id = id
        self.salary = salary

    def display_details(self):
        print(f"Name: {self.name}, ID: {self.id}, Salary:
{self.salary}")


class Manager(Employee):
    def __init__(self, name, id, salary, team_size):
        super().__init__(name, id, salary)
        self.team_size = team_size

    def conduct_meeting(self):
        print(f"{self.name} is conducting a meeting with
{self.team_size} team members.")

    def display_details(self):
        super().display_details()
        print(f"Team Size: {self.team_size}")


class Developer(Employee):
```

```
    def __init__(self, name, id, salary, programming_languages):
        super().__init__(name, id, salary)
        self.programming_languages = programming_languages

    def write_code(self):
        print(f"{self.name} is writing code in {',
'.join(self.programming_languages)}.")

    def display_details(self):
        super().display_details()
        print(f"Programming Languages: {',
'.join(self.programming_languages)}")


class Designer(Employee):
    def __init__(self, name, id, salary, design_tools):
        super().__init__(name, id, salary)
        self.design_tools = design_tools

    def create_design(self):
        print(f"{self.name} is creating a design using {',
'.join(self.design_tools)}.")

    def display_details(self):
        super().display_details()
        print(f"Design Tools: {', '.join(self.design_tools)}")


# Creating instances
employees = [
    Manager(name="Alice", id=1, salary=80000, team_size=10),
    Developer(name="Bob", id=2, salary=60000,
programming_languages=["Python", "JavaScript"]),
    Designer(name="Charlie", id=3, salary=55000,
design_tools=["Photoshop", "Illustrator"])
]

for employee in employees:
    employee.display_details()
    print()
```

```
Name: Alice, ID: 1, Salary: 80000
Team Size: 10

Name: Bob, ID: 2, Salary: 60000
Programming Languages: Python, JavaScript

Name: Charlie, ID: 3, Salary: 55000
Design Tools: Photoshop, Illustrator
```

```
class Course:
    def __init__(self, course_id, title):
        self.course_id = course_id
        self.title = title
        self.students = []

    def enroll_student(self, student_name):
        self.students.append(student_name)
        print(f"Student {student_name} enrolled in {self.title}.")

    def unenroll_student(self, student_name):
        if student_name in self.students:
            self.students.remove(student_name)
            print(f"Student {student_name} unenrolled from
{self.title}.")
        else:
            print(f"Student {student_name} is not enrolled in
{self.title}.")


class ProgrammingCourse(Course):
    def __init__(self, course_id, title, languages):
        super().__init__(course_id, title)
        self.languages = languages

    def start_project(self):
        print(f"Starting a project in {self.title} using {',
'.join(self.languages)}.")


class DesignCourse(Course):
    def __init__(self, course_id, title, software):
        super().__init__(course_id, title)
        self.software = software

    def assign_design_task(self):
        print(f"Assigning a design task in {self.title} using {',
'.join(self.software)}.")


class MathCourse(Course):
    def __init__(self, course_id, title, difficulty_level):
        super().__init__(course_id, title)
        self.difficulty_level = difficulty_level

    def assign_homework(self):
        print(f"Assigning homework in {self.title} at
{self.difficulty_level} difficulty level.")
```

```
# Demonstrating polymorphism
courses = [
    ProgrammingCourse(course_id=1, title="Intro to Python",
languages=["Python"]),
    DesignCourse(course_id=2, title="Graphic Design Basics",
software=["Photoshop"]),
    MathCourse(course_id=3, title="Advanced Calculus",
difficulty_level="High")
]

courses[0].enroll_student("Alice")
courses[1].enroll_student("Bob")
courses[2].enroll_student("Charlie")

courses[0].start_project()
courses[1].assign_design_task()
courses[2].assign_homework()

courses[0].unenroll_student("Alice")
```

```
Student Alice enrolled in Intro to Python.
Student Bob enrolled in Graphic Design Basics.
Student Charlie enrolled in Advanced Calculus.
Starting a project in Intro to Python using Python.
Assigning a design task in Graphic Design Basics using Photoshop.
Assigning homework in Advanced Calculus at High difficulty level.
Student Alice unenrolled from Intro to Python.
```

```
class Restaurant:
    def __init__(self, name, location):
        self.name = name
        self.location = location

    def open_restaurant(self):
        print(f"{self.name} at {self.location} is now OPEN.")

    def close_restaurant(self):
        print(f"{self.name} at {self.location} is now CLOSED.")


class FastFoodRestaurant(Restaurant):
    def __init__(self, name, location, drive_thru):
        super().__init__(name, location)
        self.drive_thru = drive_thru

    def serve_fast_food(self):
        print(f"{self.name} is serving fast food. Drive-thru
available: {self.drive_thru}")

    def open_restaurant(self):
        print(f"{self.name} at {self.location} is now OPEN for fast
```

```
food service.")


class FineDiningRestaurant(Restaurant):
    def __init__(self, name, location, dress_code):
        super().__init__(name, location)
        self.dress_code = dress_code

    def serve_gourmet_dishes(self):
        print(f"{self.name} is serving gourmet dishes with a dress
code: {self.dress_code}")

    def open_restaurant(self):
        print(f"{self.name} at {self.location} is now OPEN for fine
dining.")


class Cafe(Restaurant):
    def __init__(self, name, location, coffee_types):
        super().__init__(name, location)
        self.coffee_types = coffee_types

    def serve_coffee(self):
        print(f"{self.name} is serving coffee types: {',
'.join(self.coffee_types)}")

    def open_restaurant(self):
        print(f"{self.name} at {self.location} is now OPEN for coffee
lovers.")


# Demonstrating polymorphism
restaurants = [
    FastFoodRestaurant(name="Burger King", location="Downtown",
drive_thru=True),
    FineDiningRestaurant(name="Le Gourmet", location="Uptown",
dress_code="Formal"),
    Cafe(name="Coffee Corner", location="Main Street",
coffee_types=["Espresso", "Latte", "Cappuccino"])
]

for restaurant in restaurants:
    restaurant.open_restaurant()

restaurants[0].serve_fast_food()
restaurants[1].serve_gourmet_dishes()
restaurants[2].serve_coffee()

for restaurant in restaurants:
    restaurant.close_restaurant()
```

```
Burger King at Downtown is now OPEN for fast food service.
Le Gourmet at Uptown is now OPEN for fine dining.
Coffee Corner at Main Street is now OPEN for coffee lovers.
Burger King is serving fast food. Drive-thru available: True
Le Gourmet is serving gourmet dishes with a dress code: Formal
Coffee Corner is serving coffee types: Espresso, Latte, Cappuccino
Burger King at Downtown is now CLOSED.
Le Gourmet at Uptown is now CLOSED.
Coffee Corner at Main Street is now CLOSED.
```

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def display_info(self):
        print(f"Name: {self.name}, Age: {self.age}")


class Doctor(Person):
    def __init__(self, name, age, specialization):
        super().__init__(name, age)
        self.specialization = specialization

    def diagnose(self):
        print(f"Dr. {self.name} is diagnosing patients in
{self.specialization}.")

    def display_info(self):
        super().display_info()
        print(f"Specialization: {self.specialization}")


class Nurse(Person):
    def __init__(self, name, age, shift):
        super().__init__(name, age)
        self.shift = shift

    def assist(self):
        print(f"Nurse {self.name} is assisting patients during the
{self.shift} shift.")

    def display_info(self):
        super().display_info()
        print(f"Shift: {self.shift}")


class Patient(Person):
    def __init__(self, name, age, ailment):
        super().__init__(name, age)
        self.ailment = ailment
```

```
    def receive_treatment(self):
        print(f"Patient {self.name} is receiving treatment for
{self.ailment}.")

    def display_info(self):
        super().display_info()
        print(f"Ailment: {self.ailment}")


# Creating instances
people = [
    Doctor(name="Alice", age=45, specialization="Cardiology"),
    Nurse(name="Bob", age=30, shift="Night"),
    Patient(name="Charlie", age=60, ailment="Diabetes")
]

for person in people:
    person.display_info()
    print()
```

```
Name: Alice, Age: 45
Specialization: Cardiology

Name: Bob, Age: 30
Shift: Night

Name: Charlie, Age: 60
Ailment: Diabetes
```

```
class Booking:
    def __init__(self, booking_id, date):
        self.booking_id = booking_id
        self.date = date

    def confirm_booking(self):
        print(f"Booking {self.booking_id} confirmed for {self.date}.")

    def cancel_booking(self):
        print(f"Booking {self.booking_id} canceled for {self.date}.")


class HotelBooking(Booking):
    def __init__(self, booking_id, date, hotel_name):
        super().__init__(booking_id, date)
        self.hotel_name = hotel_name

    def reserve_room(self):
        print(f"Room reserved at {self.hotel_name}.")
```

```
    def confirm_booking(self):
        print(f"Hotel booking {self.booking_id} confirmed for
{self.date} at {self.hotel_name}.")


class FlightBooking(Booking):
    def __init__(self, booking_id, date, flight_number):
        super().__init__(booking_id, date)
        self.flight_number = flight_number

    def book_seat(self):
        print(f"Seat booked on flight {self.flight_number}.")

    def confirm_booking(self):
        print(f"Flight booking {self.booking_id} confirmed for
{self.date} on flight {self.flight_number}.")


class EventBooking(Booking):
    def __init__(self, booking_id, date, event_name):
        super().__init__(booking_id, date)
        self.event_name = event_name

    def reserve_ticket(self):
        print(f"Ticket reserved for {self.event_name}.")

    def confirm_booking(self):
        print(f"Event booking {self.booking_id} confirmed for
{self.date} to attend {self.event_name}.")


# Demonstrating polymorphism
bookings = [
    HotelBooking(booking_id=1, date="2023-06-16", hotel_name="Grand
Hotel"),
    FlightBooking(booking_id=2, date="2023-06-20",
flight_number="AA123"),
    EventBooking(booking_id=3, date="2023-06-25",
event_name="Concert")
]

for booking in bookings:
    booking.confirm_booking()
    booking.cancel_booking()
```

```
Hotel booking 1 confirmed for 2023-06-16 at Grand Hotel.
Booking 1 canceled for 2023-06-16.
Flight booking 2 confirmed for 2023-06-20 on flight AA123.
Booking 2 canceled for 2023-06-20.
Event booking 3 confirmed for 2023-06-25 to attend Concert.
Booking 3 canceled for 2023-06-25.
```

```python
class UniversityMember:
    def __init__(self, name, id):
        self.name = name
        self.id = id

    def show_details(self):
        print(f"Name: {self.name}, ID: {self.id}")


class Professor(UniversityMember):
    def __init__(self, name, id, department):
        super().__init__(name, id)
        self.department = department

    def teach_course(self):
        print(f"Professor {self.name} is teaching in the
{self.department} department.")

    def show_details(self):
        super().show_details()
        print(f"Department: {self.department}")


class Student(UniversityMember):
    def __init__(self, name, id, major):
        super().__init__(name, id)
        self.major = major

    def enroll_course(self):
        print(f"Student {self.name} is enrolling in courses for
{self.major} major.")

    def show_details(self):
        super().show_details()
        print(f"Major: {self.major}")


class Administrator(UniversityMember):
    def __init__(self, name, id, role):
        super().__init__(name, id)
        self.role = role

    def manage_operations(self):
        print(f"Administrator {self.name} is managing {self.role}
operations.")

    def show_details(self):
        super().show_details()
        print(f"Role: {self.role}")
```

```python
# Creating instances
members = [
    Professor(name="Dr. Smith", id=1, department="Physics"),
    Student(name="Alice", id=2, major="Computer Science"),
    Administrator(name="Mr. Brown", id=3, role="Admissions")
]

for member in members:
    member.show_details()
    print()
```

```
Name: Dr. Smith, ID: 1
Department: Physics

Name: Alice, ID: 2
Major: Computer Science

Name: Mr. Brown, ID: 3
Role: Admissions
```

```python
class Transport:
    def __init__(self, id, capacity):
        self.id = id
        self.capacity = capacity

    def start(self):
        print(f"Transport {self.id} is starting with capacity
{self.capacity}.")

    def stop(self):
        print(f"Transport {self.id} is stopping with capacity
{self.capacity}.")


class Bus(Transport):
    def __init__(self, id, capacity, route_number):
        super().__init__(id, capacity)
        self.route_number = route_number

    def pick_up_passengers(self):
        print(f"Bus {self.id} is picking up passengers on route
{self.route_number}.")

    def start(self):
        print(f"Bus {self.id} is starting on route
{self.route_number}.")


class Train(Transport):
    def __init__(self, id, capacity, number_of_coaches):
```

```
        super().__init__(id, capacity)
        self.number_of_coaches = number_of_coaches

    def depart(self):
        print(f"Train {self.id} is departing with
{self.number_of_coaches} coaches.")

    def start(self):
        print(f"Train {self.id} is starting with
{self.number_of_coaches} coaches.")


class Airplane(Transport):
    def __init__(self, id, capacity, flight_code):
        super().__init__(id, capacity)
        self.flight_code = flight_code

    def take_off(self):
        print(f"Airplane {self.id} with flight code {self.flight_code}
is taking off.")

    def start(self):
        print(f"Airplane {self.id} with flight code {self.flight_code}
is starting.")


# Demonstrating polymorphism
transports = [
    Bus(id=1, capacity=50, route_number="22B"),
    Train(id=2, capacity=200, number_of_coaches=8),
    Airplane(id=3, capacity=180, flight_code="AA101")
]

for transport in transports:
    transport.start()
    transport.stop()

Bus 1 is starting on route 22B.
Transport 1 is stopping with capacity 50.
Train 2 is starting with 8 coaches.
Transport 2 is stopping with capacity 200.
Airplane 3 with flight code AA101 is starting.
Transport 3 is stopping with capacity 180.

class Infrastructure:
    def __init__(self, location, status):
        self.location = location
        self.status = status

    def activate(self):
        self.status = "active"
```

```
        print(f"{self.__class__.__name__} at {self.location} is now
ACTIVE.")

    def deactivate(self):
        self.status = "inactive"
        print(f"{self.__class__.__name__} at {self.location} is now
INACTIVE.")


class TrafficLight(Infrastructure):
    def __init__(self, location, status, intersection_id):
        super().__init__(location, status)
        self.intersection_id = intersection_id

    def change_light(self):
        print(f"Traffic light at intersection {self.intersection_id}
is changing lights.")

    def activate(self):
        print(f"Traffic light at intersection {self.intersection_id}
is now ACTIVE.")


class StreetLight(Infrastructure):
    def __init__(self, location, status, lumens):
        super().__init__(location, status)
        self.lumens = lumens

    def adjust_brightness(self):
        print(f"Street light at {self.location} is adjusting
brightness to {self.lumens} lumens.")

    def activate(self):
        print(f"Street light at {self.location} is now ACTIVE.")


class SurveillanceCamera(Infrastructure):
    def __init__(self, location, status, resolution):
        super().__init__(location, status)
        self.resolution = resolution

    def start_recording(self):
        print(f"Surveillance camera at {self.location} is recording at
{self.resolution} resolution.")

    def activate(self):
        print(f"Surveillance camera at {self.location} is now
ACTIVE.")


# Demonstrating polymorphism
```

```
infrastructures = [
    TrafficLight(location="5th Avenue", status="inactive",
intersection_id="A1"),
    StreetLight(location="Main Street", status="inactive",
lumens=500),
    SurveillanceCamera(location="City Center", status="inactive",
resolution="4K")
]

for infra in infrastructures:
    infra.activate()
    infra.deactivate()
```

```
Traffic light at intersection A1 is now ACTIVE.
TrafficLight at 5th Avenue is now INACTIVE.
Street light at Main Street is now ACTIVE.
StreetLight at Main Street is now INACTIVE.
Surveillance camera at City Center is now ACTIVE.
SurveillanceCamera at City Center is now INACTIVE.
```

```
class GameCharacter:
    def __init__(self, name, health):
        self.name = name
        self.health = health

    def move(self):
        print(f"{self.name} is moving.")

    def attack(self):
        print(f"{self.name} is attacking.")


class Warrior(GameCharacter):
    def __init__(self, name, health, strength):
        super().__init__(name, health)
        self.strength = strength

    def use_sword(self):
        print(f"{self.name} is using a sword with strength
{self.strength}.")

    def attack(self):
        print(f"Warrior {self.name} is attacking with strength
{self.strength}.")


class Mage(GameCharacter):
    def __init__(self, name, health, mana):
        super().__init__(name, health)
        self.mana = mana
```

```
    def cast_spell(self):
        print(f"{self.name} is casting a spell with {self.mana}
mana.")

    def attack(self):
        print(f"Mage {self.name} is attacking with a spell.")


class Archer(GameCharacter):
    def __init__(self, name, health, arrows):
        super().__init__(name, health)
        self.arrows = arrows

    def shoot_arrow(self):
        print(f"{self.name} is shooting an arrow with {self.arrows}
arrows left.")

    def attack(self):
        print(f"Archer {self.name} is attacking with a bow.")


# Demonstrating polymorphism
characters = [
    Warrior(name="Thor", health=100, strength=80),
    Mage(name="Merlin", health=70, mana=150),
    Archer(name="Robin", health=85, arrows=20)
]

for character in characters:
    character.move()
    character.attack()

Thor is moving.
Warrior Thor is attacking with strength 80.
Merlin is moving.
Mage Merlin is attacking with a spell.
Robin is moving.
Archer Robin is attacking with a bow.

class Product:
    def __init__(self, product_id, name):
        self.product_id = product_id
        self.name = name

    def get_details(self):
        print(f"Product ID: {self.product_id}, Name: {self.name}")


class Electronics(Product):
    def __init__(self, product_id, name, warranty):
        super().__init__(product_id, name)
```

```
        self.warranty = warranty

    def get_warranty_info(self):
        print(f"Electronics {self.name} has a warranty of
{self.warranty} years.")

    def get_details(self):
        super().get_details()
        print(f"Warranty: {self.warranty} years")


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

    def get_size_chart(self):
        print(f"Clothing {self.name} is available in size
{self.size}.")

    def get_details(self):
        super().get_details()
        print(f"Size: {self.size}")


class Books(Product):
    def __init__(self, product_id, name, author):
        super().__init__(product_id, name)
        self.author = author

    def get_author_info(self):
        print(f"Book {self.name} is authored by {self.author}.")

    def get_details(self):
        super().get_details()
        print(f"Author: {self.author}")


# Demonstrating polymorphism
products = [
    Electronics(product_id=101, name="Smartphone", warranty=2),
    Clothing(product_id=202, name="Jeans", size="M"),
    Books(product_id=303, name="Python Programming", author="John
Doe")
]

for product in products:
    product.get_details()
    print()
```

```
Product ID: 101, Name: Smartphone
Warranty: 2 years

Product ID: 202, Name: Jeans
Size: M

Product ID: 303, Name: Python Programming
Author: John Doe
```

```python
class Asset:
    def __init__(self, asset_id, value):
        self.asset_id = asset_id
        self.value = value

    def calculate_return(self):
        print(f"Calculating return for asset {self.asset_id}.")


class Stock(Asset):
    def __init__(self, asset_id, value, ticker_symbol):
        super().__init__(asset_id, value)
        self.ticker_symbol = ticker_symbol

    def calculate_dividend(self):
        print(f"Calculating dividend for stock {self.ticker_symbol}.")

    def calculate_return(self):
        print(f"Stock {self.ticker_symbol} has a value of
{self.value}.")


class Bond(Asset):
    def __init__(self, asset_id, value, interest_rate):
        super().__init__(asset_id, value)
        self.interest_rate = interest_rate

    def calculate_interest(self):
        print(f"Calculating interest for bond with rate
{self.interest_rate}%.")

    def calculate_return(self):
        print(f"Bond with interest rate {self.interest_rate}% has a
value of {self.value}.")


class RealEstate(Asset):
    def __init__(self, asset_id, value, location):
        super().__init__(asset_id, value)
        self.location = location
```

```
    def calculate_rent(self):
        print(f"Calculating rent for property in {self.location}.")

    def calculate_return(self):
        print(f"Real estate in {self.location} has a value of
{self.value}.")


# Demonstrating polymorphism
assets = [
    Stock(asset_id=1, value=1000, ticker_symbol="AAPL"),
    Bond(asset_id=2, value=5000, interest_rate=5),
    RealEstate(asset_id=3, value=100000, location="New York")
]

for asset in assets:
    asset.calculate_return()
    print()
```

```
Stock AAPL has a value of 1000.

Bond with interest rate 5% has a value of 5000.

Real estate in New York has a value of 100000.
```