Modeling Energy Consumption with Python: Classes, Inheritance, Encapsulation, and Polymorphism

Objectives:

1.Class Definition: We will define the EnergySystem class to model the energy consumption and emission factors for different buildings.

2.Methods: We will implement methods to calculate the carbon footprint and energy savings for each building.

3.Inheritance: We will extend the base class by creating a SolarEnergySystem subclass to model solar energy production.

4.Encapsulation: We will ensure key attributes are protected to maintain controlled access to them.

5.Polymorphism: We will demonstrate polymorphism by overriding the carbon footprint calculation method in the subclass to account for solar energy production.

1: Class Definition (EnergySystem)

In [1]:
```python
# Define the EnergySystem class
class EnergySystem:
    def __init__(self, building_name, energy_consumption, emission_factor):
        # Initialize attributes
        self.building_name = building_name
        self.energy_consumption = energy_consumption  # in kWh
        self.emission_factor = emission_factor  # kg CO2 per kWh

# Example usage:
building = EnergySystem("Building A", 5000, 0.45)
print(f"{building.building_name}: {building.energy_consumption} kWh, {buildi
```

```
Building A: 5000 kWh, 0.45 kg CO2/kWh
```

This defines the EnergySystem class with attributes: building_name, energy_consumption, and emission_factor. The **init** method is the constructor that initializes the object with these attributes.

2: Methods to Calculate Carbon Footprint and Energy Savings

In [2]:
```
class EnergySystem:
    def __init__(self, building_name, energy_consumption, emission_factor):
        self.building_name = building_name
        self.energy_consumption = energy_consumption  # in kWh
        self.emission_factor = emission_factor  # kg CO2 per kWh

    # Method to calculate carbon footprint
    def calculate_carbon_footprint(self):
        return self.energy_consumption * self.emission_factor

    # Method to calculate energy savings (Assume saving 10% for now)
    def calculate_energy_savings(self):
        return self.energy_consumption * 0.10  # 10% savings

# Example usage:
building = EnergySystem("Building A", 5000, 0.45)
carbon_footprint = building.calculate_carbon_footprint()
energy_savings = building.calculate_energy_savings()

print(f"Carbon Footprint: {carbon_footprint} kg CO2")
print(f"Energy Savings: {energy_savings} kWh")
```

```
Carbon Footprint: 2250.0 kg CO2
Energy Savings: 500.0 kWh
```

calculate_carbon_footprint: Multiplies energy consumption by the emission factor to get the carbon footprint. calculate_energy_savings: Calculates 10% energy savings as an example.

3: Inheritance (SolarEnergySystem Subclass)

In [3]:
```
# Subclass SolarEnergySystem inheriting from EnergySystem
class SolarEnergySystem(EnergySystem):
    def __init__(self, building_name, energy_consumption, emission_factor, s
        # Inherit attributes from the parent class
        super().__init__(building_name, energy_consumption, emission_factor)
        self.solar_production = solar_production  # energy produced by solar

    # Method to calculate net energy consumption (energy used - solar energy
    def net_energy_consumption(self):
        return self.energy_consumption - self.solar_production

# Example usage:
solar_building = SolarEnergySystem("Solar Building", 5000, 0.45, 1500)
net_consumption = solar_building.net_energy_consumption()

print(f"Net Energy Consumption: {net_consumption} kWh")
```

```
Net Energy Consumption: 3500 kWh
```

SolarEnergySystem is a subclass that inherits from EnergySystem and adds a new attribute solar_production to represent solar panel output. The net_energy_consumption method calculates energy used minus solar energy produced.

4: Encapsulation (Protecting Attributes)

In [4]:
```
class EnergySystem:
    def __init__(self, building_name, energy_consumption, emission_factor):
        self.building_name = building_name
        self._energy_consumption = energy_consumption  # Protected attribute
        self._emission_factor = emission_factor  # Protected attribute

    # Method to access energy consumption safely
    def get_energy_consumption(self):
        return self._energy_consumption

    # Method to calculate carbon footprint
    def calculate_carbon_footprint(self):
        return self._energy_consumption * self._emission_factor

# Example usage:
building = EnergySystem("Building A", 5000, 0.45)
print(f"Energy Consumption (accessed via method): {building.get_energy_consu
```

```
Energy Consumption (accessed via method): 5000 kWh
```

The attributes energy_consumption and emission_factor are marked as protected (using _) so they can’t be directly accessed outside the class. The get_energy_consumption() method provides controlled access to the attribute.

5: Polymorphism (Overriding Methods in Subclass)

In [6]:
```python
# Subclass SolarEnergySystem overriding the carbon footprint method
class SolarEnergySystem(EnergySystem):
    def __init__(self, building_name, energy_consumption, emission_factor, s
        # Call the parent class constructor to initialize inherited attribut
        super().__init__(building_name, energy_consumption, emission_factor)
        self.solar_production = solar_production  # new attribute for solar

    # Overriding the carbon footprint method to account for solar production
    def calculate_carbon_footprint(self):
        # Access energy consumption from the parent class (inherited attribu
        net_consumption = self._energy_consumption - self.solar_production
        return net_consumption * self._emission_factor

# Example usage:
solar_building = SolarEnergySystem("Solar Building", 5000, 0.45, 1500)
carbon_footprint = solar_building.calculate_carbon_footprint()

print(f"Adjusted Carbon Footprint (after solar production): {carbon_footpri
```

```
Adjusted Carbon Footprint (after solar production): 1575.00 kg CO2
```

The **init**() method in the subclass calls super().**init**() to initialize the parent class's attributes (building_name, energy_consumption, and emission_factor).

Protected attributes: We use _energy_consumption and _emission_factor, which are protected attributes, to ensure encapsulation but allow their use within the class and subclasses.

Overriding the method: The calculate_carbon_footprint() method uses the protected attribute _energy_consumption and adjusts it by subtracting the solar production before calculating the carbon footprint.

Conclusion:

In this lab activity, we've modeled an energy system using Python's Object-Oriented Programming principles. We defined a base class for buildings' energy consumption, used methods to compute carbon footprint and energy savings, extended the class through inheritance to handle specific cases like solar energy systems, protected sensitive data using encapsulation, and demonstrated polymorphism by overriding methods in the subclass. This provides a flexible, scalable approach to managing energy data in different types of buildings.