In [1]:
```
#Smart Home Devices (Encapsulation and Property Decorators)
#Develop a system to manage smart home devices. Implement a class SmartDevice th
#● Uses encapsulation to store the device's status (__is_on, default to False).
#● Has a turn_on() and turn_off() method to change the device status.
#● Uses a property decorator to expose the device's status as a property (is_on)
#with a setter to prevent turning it on if certain conditions (like low battery)
#Task:
#1. Implement the SmartDevice class.
#2. Simulate turning the device on and off while managing conditions like low ba
#3. Use the property method to ensure users cannot turn on the device when the b

class SmartDevice:
    def __init__(self, battery_level=100):
        self.__is_on = False
        self.__battery_level = battery_level  # Battery level in percentage

    def turn_on(self):
        if self.__battery_level < 20:
            print("Cannot turn on device. Battery too low!")
        else:
            self.__is_on = True
            print("Device turned ON.")

    def turn_off(self):
        self.__is_on = False
        print("Device turned OFF.")

    @property
    def is_on(self):
        return self.__is_on

    @is_on.setter
    def is_on(self, value):
        if value:
            self.turn_on()
        else:
            self.turn_off()

    @property
    def battery_level(self):
        return self.__battery_level

    @battery_level.setter
    def battery_level(self, value):
        if 0 <= value <= 100:
            self.__battery_level = value
        else:
            print("Invalid battery level. Must be between 0 and 100.")

device = SmartDevice(battery_level=50)
device.is_on = True
print("Status:", device.is_on)

device.is_on = False
print("Status:", device.is_on)

device.battery_level = 10
```

```
device.is_on = True
print("Status:", device.is_on)
```

```
Device turned ON.
Status: True
Device turned OFF.
Status: False
Cannot turn on device. Battery too low!
Status: False
```

In [ ]: