1. Classes and Objects

Classes encapsulate data and functions that operate on the data. Objects are instances of classes.

- **Classes Defined**:
  - `Device` (abstract base class)
  - `Light` (derived class)
  - `Thermostat` (derived class)
  - `SmartHome` (manages devices)
- **Example**:
- `Light light(1); // Object of class Light`
- `SmartHome home; // Object of class SmartHome`

2. Abstraction

Abstraction hides implementation details and only shows essential features to the user.

- **Example**:
  - `Device` is an **abstract class** with pure virtual functions:
  - `virtual void turnOn() = 0;`
  - `virtual void turnOff() = 0;`
  - `virtual void displayStatus() const = 0;`
  - Users interact with `turnOn`, `turnOff`, and `displayStatus`, but the details are implemented in derived classes.

3. Inheritance

Inheritance allows a class to derive properties and behavior from another class.

- **Inheritance Hierarchy**:
  - `Device` (base class)
    - `Light` (inherits from `Device`)
    - `Thermostat` (inherits from `Device`)
- **Example**:
- `class Light : public Device {`
- `    // Light-specific attributes and methods`
- `};`

4. Polymorphism

Polymorphism allows functions to behave differently based on the object that invokes them.

- **Example**:
  - Virtual functions (`turnOn`, `turnOff`, `displayStatus`) are overridden in derived classes:
    ```
    void turnOn() override { /* Light-specific implementation */ }
    void turnOff() override { /* Thermostat-specific implementation */ }
    ```
  - **Dynamic Casting**:
    ```
    if (Light* light = dynamic_cast<Light*>(devices[i])) {
        light->adjustBrightness(brightness);
    }
    ```

5. Encapsulation

Encapsulation restricts direct access to class members and uses access specifiers (`public`, `protected`, `private`).

- **Example**:
  - `id` and `status` are `protected` members in `Device`.
  - `energyUsage` is a `const` member with a getter:
    ```
    int getEnergyUsage() const {
        return energyUsage;
    }
    ```

6. Constructor and Destructor

Constructors initialize objects, and destructors clean up resources.

- **Example**:
  - Constructor in `Device`:
    ```
    Device(int id, int energyUsage) : id(id), energyUsage(energyUsage), status(false) {}
    ```
  - Virtual destructor in `Device`:
    ```
    virtual ~Device() {
        deviceCount--;
    }
    ```

---

7. Static Members and Functions

Static members belong to the class, not to any specific object.

- **Example**:
  - Static member in `Device`:
    ```
    static int deviceCount;
    ```

    - Static function to access it:
    - `static int getDeviceCount() {`
    - `    return deviceCount;`
    - `}`

8. Constant Members

`const` members cannot be modified after initialization.

- **Example**:
- `const int energyUsage; // Initialized in the constructor`

9. Dynamic Memory Allocation

Dynamic memory allocation allows the creation of objects at runtime.

- **Example**:
- `home.addDevice(new Light(deviceId));`

10. Array of Pointers

The `SmartHome` class uses an array of pointers to manage devices.

- **Example**:
- `Device* devices[10]; // Array to store pointers to Device objects`

11. Function Overloading

Overloading allows functions with the same name to perform different tasks based on parameters.

- No direct examples in this code, but overridden virtual functions achieve similar functionality.

12. Input/Output Handling

`cin` and `cout` are used for interaction.

- **Example**:
- `cout << "Enter brightness level (0-100): ";`

- `cin >> brightness;`

Summary of Features:

Concept	Example from Code
Classes	`Device`, `Light`, `Thermostat`, `SmartHome`
Inheritance	`Light` and `Thermostat` inherit from `Device`
Polymorphism	`turnOn`, `turnOff`, `displayStatus` are overridden in derived classes
Encapsulation	Members like `id`, `status` are `protected`, accessed via member functions
Constructors	Constructors in all classes (`Device`, `Light`, `Thermostat`)
Destructor	Virtual destructor in `Device`
Static Members	`deviceCount` in `Device`
Dynamic Memory	`new Light(deviceId)`
Array of Pointers	`Device* devices[10]` in `SmartHome`