**Use Case: Adapter pattern**

Consider a home automation system that controls various smart devices. The system natively supports only Wi-Fi-enabled devices, but you also want to integrate Bluetooth devices. The Adapter Pattern allows you to connect these Bluetooth devices to the existing system.

**Working:**

**1. Interfaces:**

* **SmartHomeDevice:** Defines methods powerOn and powerOff.
* **WirelessGadget:** Defines methods connect and disconnect.

**2. Concrete Classes:**

* **SmartBulb:** Implements SmartHomeDevice and represents a Wi-Fi-connected bulb.
* **BluetoothLight:** Implements WirelessGadget and represents a Bluetooth-connected light.

**3. Adapter Class:**

* **WirelessDeviceAdapter:** Implements SmartHomeDevice and acts as an adapter between the BluetoothLight class and the SmartHomeDevice interface.
* It takes a BluetoothLight object as a constructor argument.
* It overrides the powerOn and powerOff methods of SmartHomeDevice and calls the corresponding connect and disconnect methods of the BluetoothLight object.

**4. Main Class:**

* **SmartHome:** Creates a SmartBulb object and directly calls its powerOn and powerOff() methods.
* Creates a WirelessDeviceAdapter object, passing a BluetoothLight object to its constructor. It then calls the powerOn and powerOff methods on the adapter, which in turn call the connect and disconnect methods of the BluetoothLight object.

**Benefits of using the Adapter Pattern:**

* **Reusability:** The BluetoothLight class can be reused in the context of a smart home system without modifying its code.
* **Flexibility:** New types of wireless gadgets can be easily integrated into the system by creating new adapter classes.
* **Decoupling:** The SmartHome class doesn't need to know the specific implementation of the wireless gadget. It only interacts with the SmartHomeDevice interface.