**Overview of the Observer Pattern**

The Observer pattern is a behavioral design pattern that defines a one-to-many dependency between objects. When one object (the subject) changes its state, all its dependents (observers) are notified and updated automatically. This pattern is useful for implementing distributed event handling systems.

**Code Breakdown**

1. **Subject Class**

The **Subject** class maintains a list of observers and notifies them of any state changes. Here’s how it works:

* + **Attributes**:
    - **observers**: A list to store all attached observers.
    - **state**: An integer representing the state of the subject.
  + **Methods**:
    - **getState()**: Returns the current state of the subject.
    - **setState(int state)**: Sets the state of the subject and calls **notifyAllObservers()** to inform all observers of the change.
    - **attach(Observer observer)**: Adds an observer to the list.
    - **detach(Observer observer)**: Removes an observer from the list.
    - **notifyAllObservers()**: Iterates through the list of observers and calls their **update()** method if they are active.

1. **Observer Class**

This is an abstract class that defines the interface for all observers:

* + **Attributes**:
    - **subject**: A reference to the subject that the observer is attached to.
    - **active**: A boolean flag to determine if the observer is active (default is **true**).
  + **Methods**:
    - **update()**: An abstract method that must be implemented by subclasses to define how they respond to state changes.
    - **isActive()**: Returns the active status of the observer.
    - **setActive(boolean active)**: Sets the active status of the observer.

1. **Concrete Observer Classes**

There are three concrete implementations of the **Observer** class:

* + **BinaryObserver**: Converts the state to a binary string.
  + **OctalObserver**: Converts the state to an octal string.
  + **HexaObserver**: Converts the state to a hexadecimal string.

Each of these classes:

* + Has a constructor that takes a **Subject** and attaches itself to it.
  + Implements the **update()** method to print the state in the respective number system.

1. **ObserverPatternDemo Class**

This is the main class where the program runs:

* + **Main Method**:
    - A **Subject** instance is created.
    - Three observers (**HexaObserver**, **OctalObserver**, and **BinaryObserver**) are instantiated and attached to the subject.
    - **State Changes**:
      1. The state is set to **15**. All active observers are notified and print their respective outputs.
      2. The **BinaryObserver** is deactivated by calling **setActive(false)**. This means it will not receive further updates.
      3. The state is changed to **10**. Only the active observers (**HexaObserver** and **OctalObserver**) are notified.
      4. The **BinaryObserver** is reactivated by calling **setActive(true)**.
      5. The state is changed to **5**, and all active observers are notified again.

**Output Explanation**

When the program runs, the output will show the state changes and which observers are notified:

VerifyOpen In EditorEditCopy code

1First state change: 15

2Hex String: F

3Octal String: 17

4Binary String: 1111

5BinaryObserver deactivated.

6Second state change: 10

7Hex String: A

8Octal String: 12

9BinaryObserver reactivated.

10Third state change: 5

11Hex String: 5

12Octal String: 5

13Binary String: 101

* **First State Change**: All observers print their representation of the number **15**.
* **Deactivation**: The **BinaryObserver** is deactivated, so it won't respond to the next state change.
* **Second State Change**: Only the **HexaObserver** and **OctalObserver** respond to the state change to **10**.
* **Reactivation**: The **BinaryObserver** is reactivated.
* **Third State Change**: All active observers respond to the state change to **5**.

**Summary**

This implementation effectively demonstrates the Observer pattern, allowing observers to be dynamically activated or deactivated. It ensures that only active observers are notified of state changes, enhancing flexibility and efficiency. If you have any further questions or need clarification on specific parts, feel free to ask!