The **Memento Pattern** is a behavioral design pattern used to capture and restore an object's internal state without violating encapsulation. It's often used for undo functionality in applications like text editors.

**✅ Advantages of Memento Pattern**

| **Advantage** | **Explanation** |
| --- | --- |
| **Preserves Encapsulation** | Internal state of an object is saved and restored without exposing its internal structure to other classes. |
| **Enables Undo/Redo** | Ideal for implementing rollback features, like undo in text editors, game saves, or state recovery. |
| **Easy to Implement** | Conceptually simple: create a snapshot of the object’s state and store it. |
| **Decouples State Storage** | The originator (the object whose state is saved) and caretaker (object managing the mementos) are loosely coupled. |
| **Supports Multi-level Undo** | Can store multiple mementos in a stack or list to allow stepwise undo. |

**❌ Disadvantages of Memento Pattern**

| **Disadvantage** | **Explanation** |
| --- | --- |
| **High Memory Usage** | Storing many mementos (especially if the object's state is large) consumes a lot of memory. |
| **Performance Overhead** | Creating frequent snapshots may degrade performance. |
| **Serialization Complexity** | In complex objects, storing and restoring the full state accurately may require deep copying or custom serialization logic. |
| **Careless Use Breaks Encapsulation** | If caretakers misuse or directly manipulate the memento, encapsulation can be violated. |
| **Hard to Manage Mementos** | You need to decide when to save state and how many versions to retain (e.g., how deep the undo stack should be). |

**🧠 When to Use Memento Pattern**

* Implementing **undo/redo** operations (text editors, graphic tools).
* **State rollback** in workflows (e.g., wizards, forms, games).
* **Snapshotting** objects before applying changes that might fail (e.g., transactions, simulations).