Memento Design Pattern

Memento Design Pattern, which is perfect for scenarios where you need to **capture and restore the state of an object**, like in games, editors, or undo operations.

🔐 Memento Design Pattern – Simple Definition

Memento captures and externalizes an object’s internal state so it can be restored later, without violating encapsulation.

🧠 Real-Life Analogy: "Game Save Points"

Imagine a game where Ekta reaches Level 5 and saves her game.
Later she fails in Level 6, so she "loads the save point" — this is **Memento** in action:

- Save = `createMemento()`
- Load = `restoreFromMemento(memento)`

✅ Participants in Memento Pattern

Role	Responsibility
Originator	Object whose state you want to save and restore.
Memento	Stores the internal state of the Originator.
Caretaker	Manages memento stack but never looks inside them.

✅ Java Implementation – Game Save Example

1. Memento: stores state

```
public class GameStateMemento {
    private final String level;
    private final int health;

    public GameStateMemento(String level, int health) {
        this.level = level;
        this.health = health;
    }

    public String getLevel() { return level; }
    public int getHealth() { return health; }
}
```

---

2. Originator: the game

```
public class Game {
    private String level;
    private int health;
    public void play(String level, int health) {
        this.level = level;
        this.health = health;
        System.out.println("Playing " + level + " with health: " + health);
    }

    public GameStateMemento save() {
        return new GameStateMemento(level, health);
    }

    public void restore(GameStateMemento memento) {
        this.level = memento.getLevel();
        this.health = memento.getHealth();
        System.out.println("Restored to " + level + " with health: " + health);
    }
}
```

---

3. Caretaker: maintains history (like undo/redo)

```
import java.util.Stack;

public class GameCaretaker {
    private final Stack<GameStateMemento> history = new Stack<>();

    public void save(Game game) {
        history.push(game.save());
    }

    public void undo(Game game) {
        if (!history.isEmpty()) {
            game.restore(history.pop());
        }
    }
}
```

---

4. Client Code

```
public class Main {
    public static void main(String[] args) {
        Game game = new Game();
        GameCaretaker caretaker = new GameCaretaker();

        game.play("Level 1", 100);
        caretaker.save(game);

        game.play("Level 2", 80);
        caretaker.save(game);

        game.play("Level 3", 50);
        caretaker.undo(game); // back to Level 2
        caretaker.undo(game); // back to Level 1
    }
}
```

✅ Output

```
Playing Level 1 with health: 100
Playing Level 2 with health: 80
Playing Level 3 with health: 50
Restored to Level 2 with health: 80
Restored to Level 1 with health: 100
```

✅ Where is Memento used?

Domain	Use Case Example
Text Editors	Undo/Redo of text changes
IDEs	Save points during debugging or refactoring
Games	Save game checkpoints
Forms/Wizards	Restore previous screen state
Data Entry	Revert partially-filled forms

✅ JDK Internal Example of Memento

🔹 `java.util.Date` **(legacy)**

While not a direct Memento, `clone()` of `Date` before formatting or manipulation acts like capturing a snapshot.

🔹 `Serializable`/`Externalizable`

Serialization can be used to persist object state to disk and reload — a real-world memento mechanism.

```
java
CopyEdit
ByteArrayOutputStream baos = new ByteArrayOutputStream();
ObjectOutputStream oos = new ObjectOutputStream(baos);
oos.writeObject(object);  // save

ObjectInputStream ois = new ObjectInputStream(new
ByteArrayInputStream(baos.toByteArray()));
MyObject restored = (MyObject) ois.readObject(); // restore
```

✅ Memento Pattern in Spring Boot

✅ 1. Use Case: Undo Configuration Change in Admin Panel

- **Originator**: Configuration bean (`AppConfig`)
- **Memento**: Serialized copy of the bean
- **Caretaker**: `AdminController` maintaining history stack

✅ 2. Use Case: Entity Versioning with Hibernate Envers

- Tracks entity revisions.

- Rollback to any previous version.
- Transparent snapshot mechanism.

```
java
CopyEdit
@Entity
@Audited // Hibernate Envers annotation
public class UserProfile {
    private String name;
    private String email;
}
```

✅ 3. Use Case: Workflow Engine (Camunda or Spring Statemachine)

- Save/restore workflow state (tasks, decisions, approvals).
- Ideal for long-running business processes.

✅ Summary Table

Pattern Aspect	Memento Design Pattern
Purpose	Save and restore object state
Key Concept	Encapsulation-preserving state capture
When to Use	Undo, history, checkpoints
JDK Usage	Serializable, Cloneable
Spring Boot Use Case	Config snapshots, Entity versioning, Form wizards

---