OOP Implementation Guide — Mememory Game

1. Encapsulation

Encapsulation means bundling data and the functions that work on that data inside a class, keeping internal details hidden.

How it is applied in the Mememory Game:

- The **Card** object stores its own image, state (flipped or matched), and DOM element.
    - Only the card's own methods (reveal(), hide(), lockAsMatched()) modify its appearance and state.
- The **Player** object stores player name and score.
    - Only addPoint() and reset() update the score.
- The **TurnTimer** object manages countdown internally and only exposes simple controls like start(), stop(), and reset().
- The **GameController** manages all high-level game state: current player, matched pairs, difficulty, and grid creation.

Purpose:
This keeps all variables protected inside objects and avoids global-state bugs. Each component handles only its own job.

2. Abstraction

Abstraction means exposing simple methods while hiding the complexity inside the object.

How it is applied in the Mememory Game:

- The **Card** class provides simple actions (reveal(), hide()) without showing how the DOM is structured.
- The **TurnTimer** hides the internal setInterval() logic but exposes clear events (start, stop, onTick).
- The **GameController** exposes simple functions such as startGame(), switchTurn(), and checkMatch() without forcing other parts of the code to understand shuffling, pairing, or UI updates.

Purpose:
Other parts of the game only use simple, clean methods, making the game easier to maintain and expand.

3. Inheritance

Inheritance allows a class to reuse properties and behaviors of another class and extend them with new features.

How it is applied in the Mememory Game:

- The **Card** class can be extended into specialized cards if needed (e.g., SpecialCard that triggers bonus effects, reveal cards, penalty cards).
    - It inherits all base card behaviors and adds extra abilities.
- The **Player** class can be extended into an AIPlayer later if you want single-player mode.

Purpose:
Allows the game to easily grow with new card types or advanced player logic while using the same game engine.

4. Polymorphism

Polymorphism lets different objects be used through the same interface.

How it is applied in the Mememory Game:

- The game treats **Card** and **SpecialCard** the same way.
    - The controller always calls card.reveal() or card.lockAsMatched(), and each card type handles it in its own way.
- If AIPlayer is added, both human and AI can be handled with the same player actions (e.g., player.takeTurn()).

Purpose:
The game controller does not need to know what kind of card or player it is dealing with — it just calls the same methods, allowing flexible behavior.