

## **Where did you choose \*\*safety\*\* over \*\*transparency\*\* in your Composite API and why?**

We chose safety over transparency by having the base of MenuComponent throw the UnsupportedOperationException for operations that don't make sense for a particular node type. This prevents runtime errors that could occur if clients tried to add children to leaf nodes (MenuItem) or get prices from composite nodes (Menu). While transparency would allow uniform treatment of all nodes, safety ensures compile-time or clear runtime errors when misusing the API.

## **What new behaviour becomes easy with State that was awkward with conditionals?**

The State pattern makes it easy to:

- Add new states without modifying existing conditionals
- Change transition logic in one focused location per state
- Show that each state knows only its own behaviour
- Avoid complex conditional chains that would grow too much with each new state
- Make the state machine rules explicit and self-documenting

What was previously awkward with conditionals (if/else chains checking current status and choosing valid next states) now becomes clean, encapsulated behaviour in discrete state classes.