Application or use cases of various design patterns

Reflection document

Lemichel Thibaut

GitHub Repository:
https://github.com/ThibautLemichel/FHWizardSpellManager.git

March 25, 2025

1 Description

This project is coded in java and resolves one of the most relevant problem that every wizard have : *How can they manage their spells ?*. The Wizard Spell Manager allows any wizard to add, cast and undo a spell, while taking care that most of the spell don't last forever and automaticaly undo them.
This project is based on various design patterns which garanty the maintainability and scalability of the project.

1.1 Main features

First it is possible to manage and add spells in a unique SpellBook.
Then it is possible to cast and follow the state of a spell (recovery time after casting one).
It is possible to undo a spell and some of them (FireBallSpell) have an active time, and will automatically be undo after the time is gone. Some spell (NecromancerSpell) must be undo by yourself.
Easy to manage all the spell because of the message that is written when any change in spell's state.

2 Design Patterns used

Design patterns allow developpers to code a stable, maintanable and efficient code.
Design patters used :

- Singleton pattern with SpellBook : There is only one SpellBook, that avoid redundance of the instance that is not needed. That prevents unecessary object creation and (at least a bit) increases resource efficiency.
- Factory method pattern with SpellFactory : It enhances the extensibility and the flexibility for creating a spell objects, it allows easy addition of new spell types without modifying existing logic.
- State pattern with SpellState : provides a clear logic of spell lifecycle, specially the transition between Charged, Cast and Recovery state. That organize the logic of spell behavior.
- Command pattern with CastSpellCommand : enable action reversibility and structured spell invocation.

- Observer pattern with SpellStatusObserver : real-time notification on spell state changes. This provide valuable debugging information and enhances the observability of the spell management s

All combined they make the code more readable and more reusable. Because of the flexible, it is easy to add new features, to add new spell mechanics, adjust spell properties or to add new fonctionnalities like mana consumption. It ensures to have a better overview to debug the code while each state changement is notified thanks to the observer.

3 Challenges faced during development

First I had to be careful when integrating State and Observer Patterns to ensure that is the state changes is triggered without introducing circular dependencies or redundant event calls. To do so I centralized the observer in SpellBook.

I wanted that spells have different recovery times so I've created TimedSpell an interface that standardizes spells active duration and RecoveryState.

Some spells can have time duration. They require careful reversion. Immediately undoing a spell must account for potential state changes. I've implemented state resets using undo operations to ensure the spell accurately returns to its ChargedState without disrupting timers.