Abstract Factory Patterns in our Property Management System

Introduction

We need to identify design patterns that can be applied to our property management system. After analyzing our current class diagram, we've identified two places where the Abstract Factory pattern would be really useful. This document explains why we chose these patterns and how they would help improve our project.

What is an Abstract Factory Pattern?

The Abstract Factory is a design pattern that helps us create groups of related objects without having to know their exact types in advance. In other words, it is like a factory that makes other factories. This pattern is very helpful when we want our system to work without depending on the details of how its products are made or structured. Using Abstract Factory makes our code more flexible and easier to expand in the future.

Pattern 1: Report Factory

Description

The Report Factory would be responsible for creating different types of reports in our property management system. Looking at our current class diagram, we already have a Report class, but it doesn't have a good way to handle different types of reports that different users might need.

Why this pattern is needed

Our system serves multiple user roles, each with distinct reporting requirements, such as:

- Financial reports (for Finance Officers)
- Maintenance reports (for Property Managers)
- Occupancy reports (for Administrators)
- Tenant history reports (for Support Staff)

At present, the Report class only covers common attributes such as:

- reportID
- reportType
- generatedDate
- generatedBy
- content

However, it does not explain how to create different types of reports, each with their own data and formatting needs. Without a factory, we would have to use a lot of if-else statements in different places in our code, which would make the system harder to maintain and update. Adding a new type of report, like a tax report, would also mean changing code in many different parts of the project.

How the Report Factory would work

To solve these problems, we suggest adding an abstract ReportFactory interface with a method like createReport(). We would then have different factory classes, such as:

- StandardReportFactory: for making basic reports
- SpecializedReportFactory: for making more complex reports that might include charts or analysis

Each factory would create the right type of report, like StandardReport or SpecializedReport. For example, if a Finance Officer needs a detailed financial report, the system would use the SpecializedReportFactory to produce a report with all the necessary analysis and projections.

Pattern 2: Legal Document Factory

Description

This pattern addresses the need to generate various types of legal documents within our property management system. While our current class diagram includes a LegalAdvisor class and a LeaseContract class, it does not give us a flexible way to create all the different legal documents our system might need.

Why this pattern is needed

Our system must support the creation of numerous legal documents, including:

- Lease agreements
- Eviction notices
- Property inspection reports

- Maintenance agreements
- Disclosure documents
- Custom terms agreements

While the class diagram shows LegalAdvisor methods related to LeaseContract and defines a CustomTerms class, it does not have a single, clear way to create all the legal documents we need, each with their own format and rules. Legal documents can be different based on:

- The document type (for example, lease, notice, or disclosure)
- The property type (residential or commercial)
- The regional legal requirements (different areas have different laws)

Without a central factory, the code for making these documents would be repeated in many places, making the system harder to maintain and increasing the chance of legal errors.

How the Legal Document Factory would work

We suggest adding an abstract LegalDocumentFactory interface with methods such as:

- createLeaseContract(data)
- createVerificationReport(data)

Concrete versions of this factory could include:

- StandardLegalDocumentFactory: For creating standard legal documents
- ComplexLegalDocumentFactory: For creating more complex legal documents with additional terms or requirements

Each factory would create document objects that use the common LegalDocument interface. This makes sure all documents follow the same standard, but also allows for differences based on how complex the document needs to be. For example, when a new tenant signs a lease, the system would use either the standard or complex factory—depending on the situation—to create all the needed documents with the right terms and legal information.

Benefits for Our Project

Implementing these two Abstract Factory patterns will bring several important benefits:

- **Cleaner and easier-to-maintain code:** Having one place for creating reports and documents makes the code less messy and easier to update.
- **Easy to add new types:** We can add new report or document types without changing

existing code.

- **Consistency:** All reports and legal documents will follow the same rules and format, so there will be fewer mistakes.
- **Lower legal risk:** By creating legal documents in one central way, we are less likely to miss important requirements.
- **Adaptability:** The system can more easily handle changes in laws or business rules.
- **Separation of concerns:** The way documents are created is kept separate from other business logic, making the system more organized and easier to test.
- **Ready for the future:** The system is built to handle new needs and grow with the business.

Conclusion

By looking at our current class diagram, we found two important areas where we can use the Abstract Factory pattern: creating reports and making legal documents. Adding these patterns will help us fix real issues in our design and will make our property management system easier to maintain, easier to expand, and stronger for the future.