Factory Method pattern 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 Factory method pattern would be really useful. This document explains why we chose these patterns and how they would help improve our project.

What is a Factory Method Pattern?

The Factory Method is a creational pattern that defines an interface for creating a single object, but lets subclasses decide which concrete class to instantiate. In other words, you call a “factory” method on a base class or interface, and concrete subclasses override that method to produce the exact product you need. This decouples your code from concrete types, making it easy to introduce new products simply by writing a new subclass that implements the factory method—no changes to existing client code required.

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.

Benefits for Our Project

Implementing this Factory method pattern 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 an important area where we can use the Factory method pattern: creating reports. Adding this pattern 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.