Documentação e Explicação do Sistema

Jonatas Kalebe Machado de Freitas, Pedro Alano Vieira da Silva Portela

Contents

Introdução

Este documento apresenta o design, a implementação e a documentação de um sistema de gerenciamento de cursos, empregando conceitos fundamentais de orientação a objetos, padrões de projeto GoF, manipulação de arquivos, testes, interface gráfica, threads e UML. Foram utilizados pelo menos três padrões de projeto GoF: o **Factory Method**, o **Observer** e o **Iterator**. Além disso, foi elaborado um projeto em UML, usando diagramas relevantes para justificar as decisões de design. O código foi implementado em Java, incluindo conceitos de classes, interfaces, encapsulamento, associações, herança, polimorfismo, manipulação de arquivos, tratamento de exceções, testes unitários, interface gráfica e threads.

1 Padrões de Projeto Utilizados

1.1 Factory Method

O padrão *Factory Method* foi empregado para a criação de objetos do tipo `Course`. A classe abstrata `CourseFactory` define a interface para criação de cursos, e a classe concreta `DefaultCourseFactory` implementa o método de criação. Isso promove a flexibilidade na criação de cursos, permitindo a substituição da fábrica sem alterar o código cliente. Esse padrão foi um dos quatro padrões apresentados pelo professor em sala de aula.

1.2 Observer

O padrão *Observer* foi utilizado para manter a interface gráfica e outros componentes atualizados quando há mudanças no estado do sistema. A interface `Subject`, implementada por `CourseManager`, gerencia uma lista de observadores (`Observer`) que são notificados sempre que há alterações nos dados. Assim, quando um novo curso é criado ou um aluno é matriculado, todos os observadores são notificados para refletir imediatamente as mudanças. Este padrão foi o padrão apresentado pelo grupo no seminário.

1.3 Iterator

O padrão *Iterator* foi aplicado para percorrer coleções de cursos. A classe `CourseCollection` utiliza `CourseIterator` para fornecer uma maneira uniforme de acessar os elementos, sem expor sua representação interna. Esse padrão torna a iteração sobre coleções mais flexível e desacoplada da implementação real da lista de cursos.

2 UML e Justificativas de Design

Foi elaborado um diagrama de classes UML para representar a estrutura do sistema, incluindo as classes `Course`, `Student`, `Professor`, `CourseManager`, `CourseFactory`, `DefaultCourseFactory`, `CourseCollection`, `CourseIterator`, `Observer`, `Subject` e as classes relacionadas à interface gráfica e às threads. O diagrama de classes foi escolhido por evidenciar as relações de herança, composição, associação e implementação de interfaces entre as classes, tornando claras as aplicações dos padrões de projeto e do paradigma orientado a objetos.

Também poderiam ser usados diagramas de sequência para demonstrar a interação entre objetos durante operações como a criação de um curso e a

notificação de observadores, porém, optou-se principalmente pelo diagrama de classes pela ênfase na estrutura e na aplicação dos padrões.

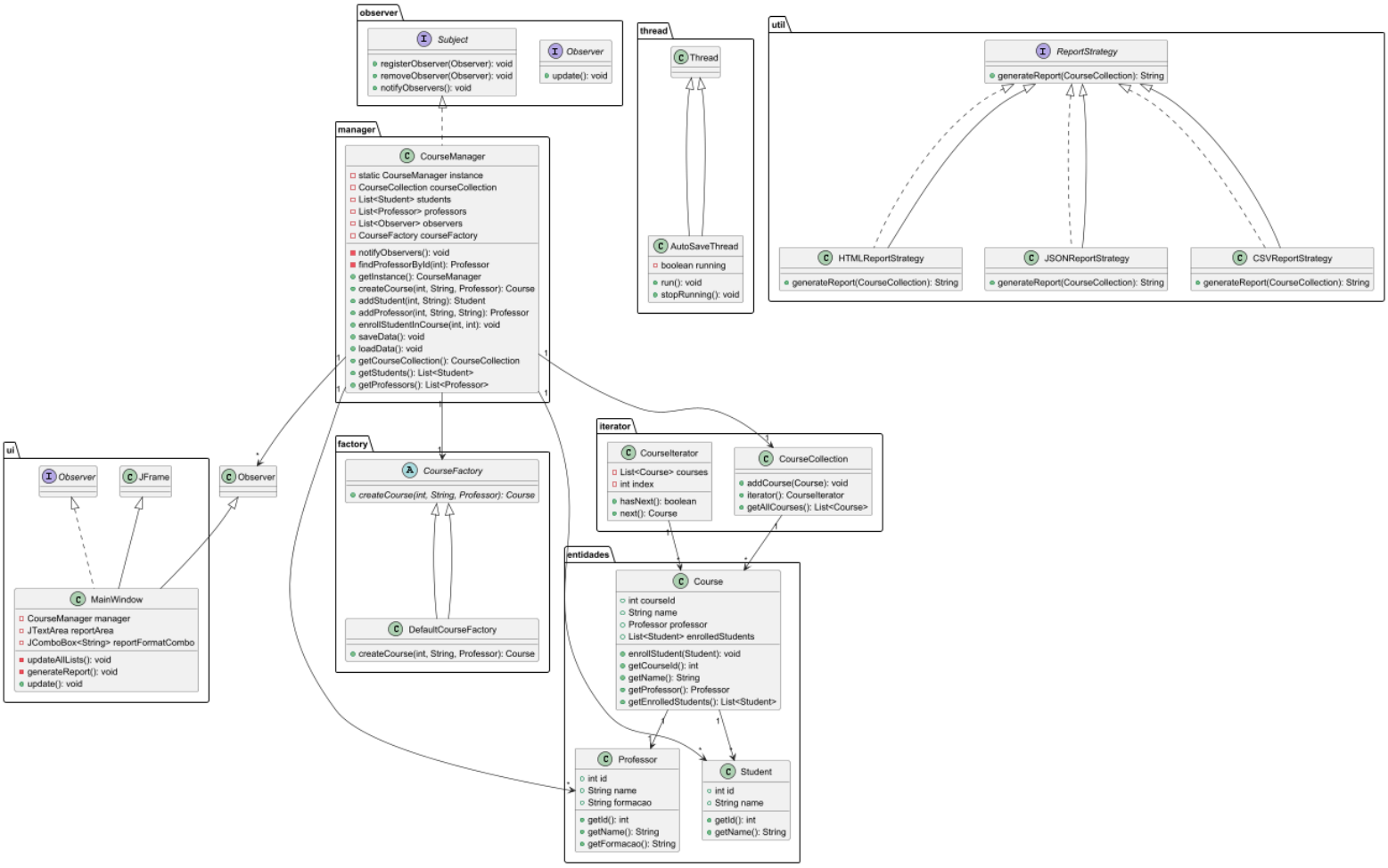

Figure 1: Diagrama de classes UML do sistema

3 Conceitos Fundamentais

Foram empregados conceitos de classes, interfaces, encapsulamento, herança e polimorfismo. Por exemplo, a classe `Course` representa a entidade curso com atributos e métodos específicos, enquanto a interface `Observer` permite o polimorfismo de diferentes tipos de observadores. O `CourseManager` faz uso de encapsulamento para proteger o estado interno. A herança é utilizada na implementação do padrão *Factory Method*, onde `DefaultCourseFactory` herda de `CourseFactory`.

4 Conceitos Avançados

4.1 Manipulação de Arquivos

A classe `CourseManager` lê e grava dados em arquivo (`data.txt`) usando `BufferedReader` e `BufferedWriter`. Esse recurso permite a persistência dos dados de cursos, professores e alunos, bem como as matrículas realizadas.

4.2 Tratamento de Exceções

O sistema implementa controle robusto de exceções utilizando `try-catch` para capturar e tratar falhas como erros de leitura ou gravação em arquivos e inconsistências nos dados. Por exemplo:

- Durante a leitura dos dados do arquivo (`data.txt`), o sistema valida os dados e ignora entradas malformadas, evitando que informações incorretas comprometam o estado do programa.
- Caso ocorra uma falha durante a gravação, uma mensagem informativa é registrada e o programa tenta reestabelecer o estado inicial.

Esse controle garante que o sistema funcione de maneira contínua mesmo diante de falhas inesperadas, protegendo tanto a integridade dos dados quanto a experiência do usuário.

4.3 Testes

Foram desenvolvidos testes unitários utilizando JUnit (`CourseManagerTest`) para verificar a criação de cursos e a matrícula de alunos, assegurando a qualidade do sistema.

4.4 Interface Gráfica

Uma interface gráfica Swing (`MainWindow`) foi implementada para interagir com o usuário de forma amigável. É possível adicionar professores, cursos, alunos, efetuar matrículas e gerar relatórios.

4.5 Threads

A `AutoSaveThread` executa periodicamente a operação de salvamento dos dados, garantindo que as informações estejam sempre atualizadas em arquivo sem bloquear a interface do usuário.

5 Documentação do Código e Relação com os Padrões

No código, cada classe e interface é documentada com comentários explicando sua função, a aplicação do padrão de projeto e a razão de sua existência. Por exemplo, `CourseFactory` e `DefaultCourseFactory` são documentadas indicando sua função na criação de cursos, `CourseManager` descreve como o padrão *Observer* é aplicado, e `CourseCollection` com `CourseIterator` esclarece o padrão *Iterator*.

Os padrões de projeto agregam valor ao sistema, tornando-o mais flexível, escalável e fácil de manter. A fábrica abstrai a criação de objetos, o observador mantém a interface sincronizada com os dados, e o iterador padroniza o acesso à coleção de cursos.

Conclusão

A combinação dos padrões de projeto, conceitos fundamentais de orientação a objetos, manipulação de arquivos, controle de exceções, testes, interface gráfica e threads resultou em um sistema robusto e bem estruturado. A UML auxiliou na visualização do design, justificando as escolhas e facilitando o entendimento da arquitetura global.