UNIVERSIDADE FEDERAL DE ALAGOAS — UFAL

Campus A. C. Simões

Ciência da Computação

Walber Luis Santos da Paixão

RELATÓRIO DE IMPLEMENTAÇÃO TÉCNICA — SISTEMA DE REDE SOCIAL JACKUT

Maceió — AL

2025

Walber Luis Santos da Paixão

RELATÓRIO DE IMPLEMENTAÇÃO TÉCNICA — SISTEMA DE REDE SOCIAL JACKUT

Trabalho apresentado como pontuação parcial da AB2 do 4° semestre da disciplina de Programação Orientada a Objetos

Professor: Mario Hozano

Maceió — AL

2025

1. Introdução

Este projeto demonstra a aplicação prática de conceitos avançados de OO, como encapsulamento, herança, polimorfismo e princípios SOLID, além de padrões como Strategy, Observer e Facade. O código foi desenvolvido visando baixo acoplamento, alta coesão e facilidade de manutenção, garantindo que novas funcionalidades possam ser adicionadas com impacto mínimo na estrutura existente.

Os testes automatizados (como os arquivos us5_1.txt a us9_2.txt) foram essenciais para validar o comportamento do sistema e garantir a integridade dos dados, especialmente em operações críticas como remoção de usuários e persistência de relacionamentos.

A seguir, detalhamos cada aspecto do projeto, incluindo decisões de design, implementação e resultados obtidos.

2. Design do Sistema

2.1 Diagrama de Classes

https://github.com/walberluis/Jackut/blob/main/Diagrama-Jackut-Walber.png

Observação: Não foi possível incluir a imagem diretamente aqui porque o arquivo PNG do diagrama de classes era muito grande. Em vez disso, disponibilizei um link para o repositório no GitHub onde a imagem está hospedada.

2.2 Padrões de Projeto Utilizados

- *Facade*: Implementado na classe Facade que fornece uma interface simplificada para todas as operações do sistema, escondendo a complexidade das classes internas.
- *Strategy*: Utilizado no sistema de notificações através da interface NotificacaoStrategy e sua implementação ConsoleNotificacao, permitindo flexibilidade na forma como as notificações são enviadas.
- *Observer*: Embora não implementado explicitamente, o conceito é utilizado no sistema de mensagens e recados, onde os usuários são notificados quando recebem novas mensagens.
- *Singleton*: Utilizado implicitamente no sistema, garantindo que haja apenas uma instância das classes principais como Sistema.

2.3 Justificativas de Design

- *Separação de Responsabilidades*: O sistema foi dividido em camadas claras (Fachada, Sistema, Domínio) para facilitar a manutenção e extensão.
- *Baixo Acoplamento*: As classes foram projetadas para dependerem de interfaces e não de implementações concretas, como no caso do sistema de notificações.
- *Alta Coesão*: Cada classe tem uma responsabilidade bem definida, seguindo o princípio da responsabilidade única.
- *Serialização*: Todas as classes de domínio implementam Serializable para permitir persistência dos dados.

3. Relatório das User Stories Implementadas (US5-US9)

User Story 5 - Criação de Comunidades

Implementação:

- Criada a classe Comunidade com atributos: nome, descrição, dono e membros
- Implementados métodos no Sistema:
    1. criarComunidade();
    2. getDescricaoComunidade();
    3. getDonoComunidade();
    4. getMembrosComunidade()

Padrões Utilizados:

- Singleton (implícito) para garantir unicidade no nome das comunidades
- Factory Method na criação de comunidades

Dificuldades:

- Garantir a unicidade dos nomes de comunidades
- Manter consistência quando o dono é removido do sistema

Testes Realizados:

- Criação de comunidades com nomes únicos
- Verificação de dono e membros iniciais
- Tentativas de criar comunidades com nomes duplicados
- Persistência dos dados após reinicialização

User Story 6 - Adição a Comunidades

Implementação:

- Adicionado método adicionarComunidade() no Sistema
- Atualização dos conjuntos de membros nas comunidades e de comunidades nos usuários
- Validação de existência da comunidade; Usuário já ser membro; Sessão válida

Padrões Utilizados:

- Observer para notificar membros sobre novas adições (se necessário)
- Mediator através da classe Sistema que coordena a interação

Dificuldades:

- Manter consistência entre as listas de membros e comunidades do usuário
- Ordenação alfabética na exibição de membros

Testes Realizados:

- Adição válida de usuários a comunidades
- Tentativas de adicionar a comunidades inexistentes
- Verificação de listagem ordenada de membros
- Persistência das relações

User Story 7 - Envio de Mensagens a Comunidades

Implementação:

- Criada classe Mensagem com remetente, comunidade e texto
- Implementados métodos:
    1. enviarMensagem() - Distribui para todos os membros
    2. lerMensagem() - Consome mensagens não lidas
    3. Diferenciado claramente de recados (mensagens privadas)

Padrões Utilizados:

- Observer para notificar membros sobre novas mensagens
- Strategy no sistema de notificações

Dificuldades:

- Garantir entrega ordenada das mensagens
- Diferenciar claramente do sistema de recados existente

Testes Realizados:

- Envio para comunidades com um e múltiplos membros
- Verificação de ordem de recebimento
- Mensagens de diferentes remetentes
- Persistência das mensagens não lidas

User Story 8 - Novos Relacionamentos

Implementação:

- Fã-Ídolo:
    1. Adicionados conjuntos idolos e fas na classe Usuario
    2. Métodos adicionarIdolo(), ehFa(), getFas()
- Paquera:
    1. paqueras e métodos relacionados
    2. Lógica de notificação mútua automática
- Inimizade:
    1. Conjunto inimigos e bloqueio de interações
    2. Restrições em amizades, recados e outros relacionamentos

Padrões Utilizados:

- State para diferentes estados de relacionamento
- Observer nas notificações de paqueras mútuas

Dificuldades:

- Complexidade nas restrições de inimizade
- Garantir consistência nos relacionamentos bidirecionais
- Notificação automática de paqueras mútuas

Testes Realizados:

- Relacionamentos unidirecionais e mútuos
- Bloqueios por inimizade
- Notificações automáticas

User Story 9 - Remoção de Conta

Implementação:

- Método removerUsuario() no Sistema que:
    1. Remove de todas as comunidades
    2. Apaga comunidades onde era dono
    3. Limpa referências em relacionamentos
    4. Remove recados e mensagens associadas
    5. Encerra sessões ativas

Padrões Utilizados:

- Facade para simplificar a complexidade da operação
- Observer para notificar sobre a remoção (se necessário)

Dificuldades:

- Garantir limpeza completa de todas as referências
- Manter consistência após remoção
- Performance em sistemas com muitos usuários

Testes Realizados:

- Remoção com comunidades, relacionamentos e mensagens
- Verificação de limpeza de referências
- Persistência das alterações
- Tentativas de acesso após remoção

4. Conclusão

O projeto Jackut implementa com sucesso os conceitos de Orientação a Objetos, demonstrando:

- Encapsulamento: Dados e comportamentos estão adequadamente encapsulados nas classes.
- Herança: Utilizada de forma moderada, apenas quando realmente necessário.
- Polimorfismo: Principalmente através da interface de notificações.
- Princípios SOLID: Especialmente Single Responsibility e Dependency Inversion.

As principais lições aprendidas foram:

- A importância de um design bem pensado desde o início para facilitar a adição de novas funcionalidades.
- Como padrões de projeto podem resolver problemas comuns de forma elegante.
- A necessidade de balancear entre flexibilidade e complexidade.