UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL
BACHARELADO EM TECNOLOGIA DA INFORMAÇÃO
IMD0410 - PROGRAMAÇÃO CONCORRENTE E DISTRIBUÍDA

Trabalho da 2ª Unidade: Sincronização em programas concorrentes
Lead Me

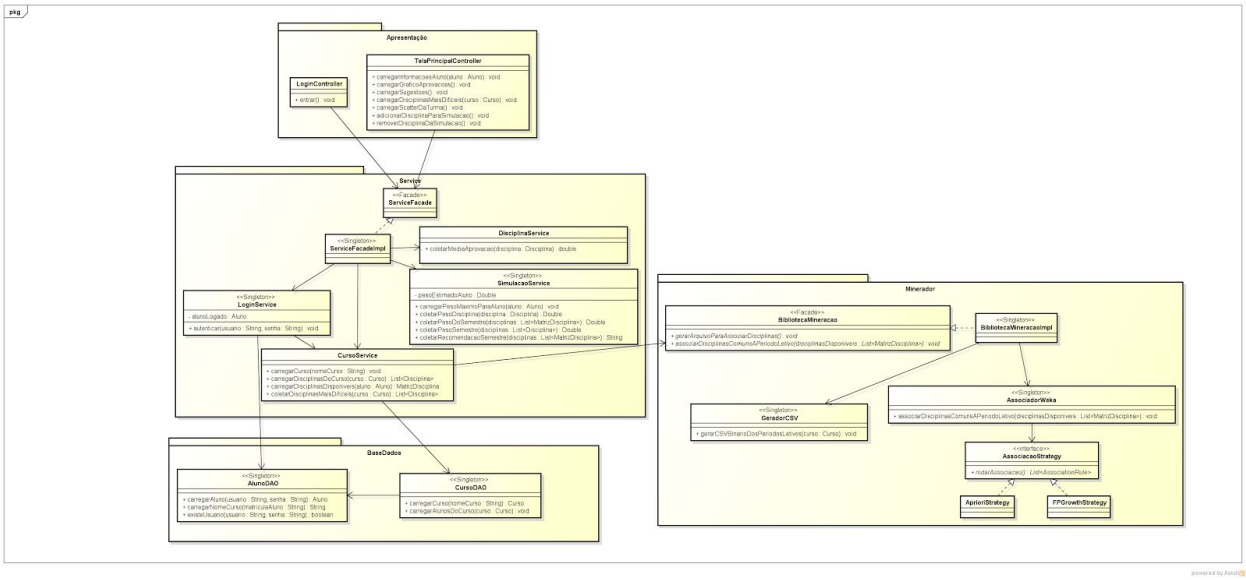
Discente: Rafael Oliveira Mendes

1. O PROBLEMA ABORDADO

Na universidade, um problema recorrente do aluno é a dificuldade de planejar o semestre que está para começar, em relação à quantas e quais disciplinas deve cursar. Isso se dá pelo fato do aluno não ter acesso direto à informações estatísticas sobre taxas de aprovação, ou quais disciplinas costumam ser pagas no mesmo semestre com sucesso. A falta de informações e orientações pode causar escolhas equivocadas e por consequência, um semestre complicado e frustrante.

2. COMO A SOLUÇÃO FOI PROJETADA

O sistema realiza análises estatísticas, além de utilizar mineração de dados para levantar informações relevantes, como média de aprovação das disciplinas, associar disciplinas de acordo com os históricos dos alunos, calcular dificuldade de um semestre que o aluno seleciona para simular, entre outras funcionalidades.
O sistema foi feito em ambiente desktop, no formato de camadas. Abaixo temos o diagrama de classes do sistema.



Temos a camada de apresentação, composta por classes para controlar as telas do sistema. Essas classes utilizam os serviços da camada Service, através da fachada. As funções envolvendo a coleta de dados fazem chamada às classes DAO, que fica na camada de dados. Análises das disciplinas relacionadas são feitas através da camada biblioteca de mineração, também acessada através de uma fachada. A projeção da solução concorrente está na camada de dados, onde há a coleta dos dados do curso do aluno logado, onde são coletadas as matrizes curriculares, os alunos do curso, e

com isso, as turmas existentes ao longo dos períodos letivos. Uma vez as matrizes curriculares são independentes entre si, a sua coleta é feita concorrentemente. Além disso, quando um curso é carregado, os seus alunos também são, e novamente, os alunos são independentes entre si, e assim, são carregados também de forma concorrente.

3. LÓGICA DA SINCRONIZAÇÃO

Os recursos compartilhados são os objetos do tipo Curso, Disciplina e Turma. A solução proposta foi criar uma classe chamada RecursoCompartilhado (encapsulando o bloqueio explícito), e qualquer operação que necessite ser realizada em exclusão mútua, o objeto recurso é chamado para obter acesso à região crítica, e após a sua conclusão, o recurso é liberado. Logo, as classes Curso, Disciplina e Turma possuem um atributo chamado recurso, para sincronizar as operações concorrentes.
No caso do curso, o objeto é compartilhado no momento em que as matrizes curriculares são carregadas, e cada thread é responsável por carregar uma matriz curricular e adicioná-la na lista de matrizes do objeto curso (classe CarregadorMatrizCurricular).
No caso da disciplina, é compartilhada no momento em que as turmas dessa disciplina são criadas. Isso acontece quando o sistema carrega os alunos, sendo que cada aluno é carregado por uma thread. Quando a thread lê o histórico do aluno e verifica uma turma em que participou, ele verifica no objeto disciplina se esse objeto turma já existe, e caso não exista, é criada a instância da turma e adicionada na lista de turmas da disciplina. A operação de adicionar uma turma na disciplina deve ser feita em exclusão mútua, e por isso o objeto recurso é chamado para a thread adquirir acesso à região crítica.
Para o objeto turma, a região crítica acontece quando a thread que está carregando um determinado aluno vai adicioná-lo à lista de alunos da turma. Nesse caso novamente o objeto recurso é chamado para adquirir o acesso à região crítica.

Obs.: Média de execução concorrente:55,06 ms
Média sequencial:102,06 ms

4. CORRETUDE DA SOLUÇÃO

Conforme a solução descrita no item anterior, foi criado um mecanismo de sincronização para as classes Curso, Disciplina e Turma, os quais são os recursos compartilhados da aplicação. Os métodos que são chamados por diferentes threads são cobertos pelas chamadas pelo objeto recurso para adquirir acesso à região crítica,

e isso garante a exclusão mútua. Logo após a cada acesso à região crítica, o recurso é liberado também utilizando o objeto recurso.
É garantida a ausência de deadlock pelo fato de não existir dependência circular (nas diferentes situações de concorrência, existia disputa por apenas um recurso compartilhado, e assim, não é possível existir dependência circular).
É garantida também a ausência de starvation, pois o mecanismo utilizado é o bloqueio explícito (lock), o qual foi instanciado com a flag fair = true, e isso quer dizer que cada thread que procura acesso à região crítica e é bloqueado, entra em uma fila de espera, e no momento em que o recurso é liberado, é chamado na ordem da fila, garantindo assim, a justiça na execução com as diversas threads.

5. DIFICULDADES ENCONTRADAS

Esse projeto é naturalmente sobre mineração de dados, e a principal dificuldade encontrada é justamente a ausência desses dados. Isso implicou a simulação dos dados para construção e teste do sistema. Análises de desempenho da solução concorrente em relação à sequencial foram prejudicadas em razão do tamanho dos dados coletados (a base é pequena demais para que a solução concorrente se destacasse, apesar de já exibir números mais favoráveis).

6. INSTRUÇÕES DE COMPILAÇÃO E EXECUÇÃO

Projeto criado no Netbeans: Maven > Aplicação JavaFX
Após download e extração do arquivo zip, abra o NetBeans, e clique em Arquivo > Abrir Projeto.

Atualize a referência para a biblioteca controlsfx-8.40.13.jar para apontar para o jar que está dentro da pasta do projeto, na opção Dependências, clicando com o botão direito e selecionando a opção Instalar artefato manualmente.

Execute o projeto clicando em Executar projeto (ou apertando F6).

O sistema de autenticação está simulado, e um login possível para entrar é usuário: 20172017 e senha: 123456.

Link do repositório: https://github.com/Rafael260/Lead-Me-Maven