Programação Modular
Trabalho Prático 1:
Rede Social de Pesquisadores

Sandro Miccoli - 2009052409 - smiccoli@dcc.ufmg.br
Frederico Figueiredo - 2010054371 - fredfig@dcc.ufmg.br

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)

15 de abril de 2015

Resumo. *Esse relatório descreve como foi implementado o problema proposto no Trabal ho Prático 1: uma Rede Social de Pesquisadores. A Seção 1 introduz o problema proposto e dá uma visão geral da solução implementada. Cada seção irá descrever detalhes do sistema desenvolvido, abrangendo desde o planejamento (Seção 2), as decisões de implementação (Seção 3) e os testes realizados (Seção 5). Finalmente concluímos (Seção 6) a documentação com reflexões sobre o aprendizado durante a execução do trabalho.*

1. Introdução

O problema proposto neste trabalho foi a implementação de uma rede social (fictícia) de pesquisadores. O objetivo principal desta rede social é armazenar dados sobre pesquisadores, artigos e veículos de publicação. A proposta do trabalho prático é de extrair informações sobre este conjunto complexo de dados. Informações do tipo: popularidade de cada pesquisador; fator de impacto de cada veículo de publicação; e, finalmente, a qualidade de cada artigo.

O programa implementado funciona será detalhado superficialmente aqui, porém a Seção 3 irá explicar em profundo os detalhes de implementação. Vários arquivos de entrada são passado com os dados brutos da rede social. Inicialmente o programa registra todos os pesquisadores, veículos de publicação e artigos. Após isso é feito uma relação entre artigos e pesquisadores, como por exemplo, quem é autor de qual artigo e qual a sua posição de autoria neste artigo específico. Os artigos são, então, verificados para que possam ser computadas a quantidade de citações de cada artigo. Essa parte inicial simplesmente executa a preparação dos dados que serão, posteriormente, utilizados para calcular as métricas e gerar informações interessantes sobre a rede social.

2. Planejamento

O planejamento que antecedeu à implementação ocorreu da seguinte forma: estudamos o problema e inicialmente geramos diagramas UML para melhor compreensão do que iríamos desenvolver. Uma situação inesperada durante esse planejamento foi a percepção de que não precisaríamos do arquivo de entrada Grafo Pesquisadores que foi passado junto com a documentação do trabalho prático. Essa entrada se tornou obsoleta pois o arquivo de entrada Grafo Bipartido, o qual indica pesquisadores, artigos e sua respectiva ordem de autoria, contém a informação que seria passada pelo primeiro arquivo. Além disso criamos um repositório git online para que pudéssemos trabalhar em paralelo no mesmo código e caminhar rápido com o trabalho.

3. Implementação

Após gerar o diagrama UML que indicava quais classes precisariam ser implementadas, foi definida como seria montada a estrutura de dados do programa. Abaixo, na Figura 1, é possível ver uma versão simplificada do diagrama de classes do sistema implementado. Simplificada pois não contém nenhuma informação de atributos ou métodos, apenas de relacionamento entre classes e pacotes. Isso foi feito para não poluir a documentação com um diagrama tão complexo. Ao final deste documento é possível ver o diagrama de classes completo (Figura 2).

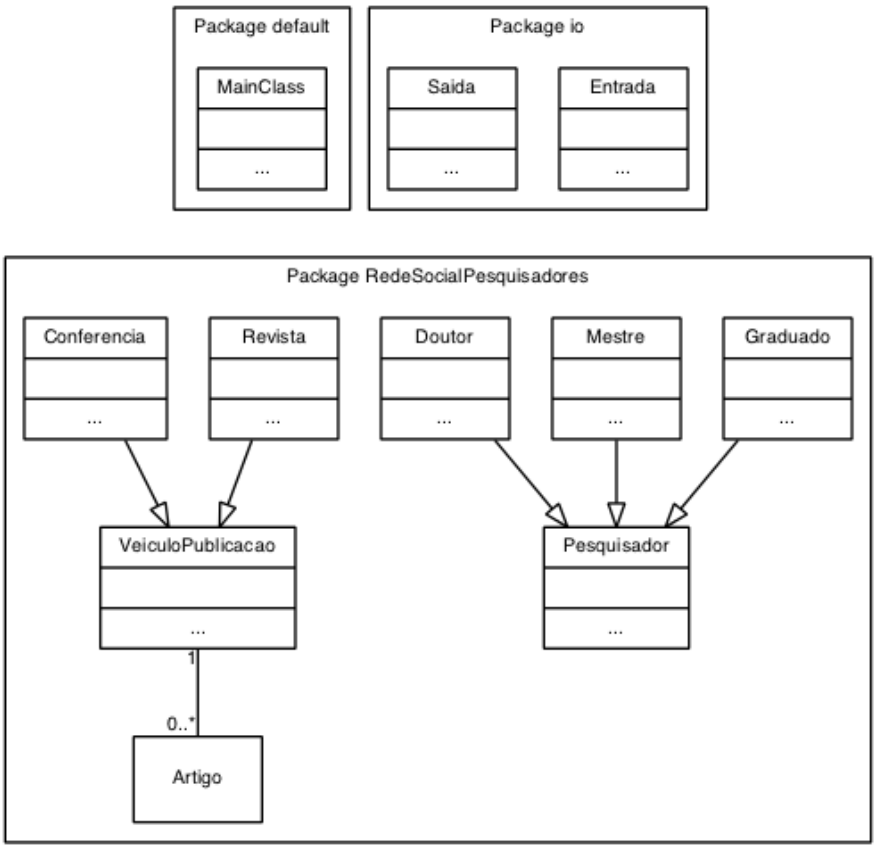

Figura 1. Diagrama de classes simplificado

Em resumo, as seguintes classes e subclasses foram criadas:

4. Classes implementadas

Pesquisador

Classe que contém os dados essenciais de um pesquisador e seus métodos padrões, como *getters*, *setters* além de implementações gerais de certas métricas, como o cálculo de popularidade.

Graduado

Classe filha de Pesquisador. Apenas sobrescreve o método para o cálculo de popularidade.

Mestre
: Classe filha de Pesquisador. Sobrescreve método de impressão e de popularidade. Adiciona possibilidade de orientar alunos de graduação.

Doutor
: Classe filha de Pesquisador. Sobrescreve método de impressão e de popularidade. Permite orientar alunos de graduação, mestrado e doutorado, então vários métodos *getters* e *setters* foram implementados.

Veículo Publicação
: Classe que possui dados e métodos padrões de tipos diferentes de veículos de publicação.

Revista
: Classe filha de Veículo Publicação. Sobrescreve apenas o método de cálculo de valor de impacto.

Conferência
: Classe filha de Veículo Publicação. Sobrescreve apenas o método de cálculo de valor de impacto.

Artigo
: Classe que representa artigos, todos seus dados, como, por exemplo, a quantidade de citações que cada um possui.

Entrada
: Classe que lida com leitura de arquivos.

Saída
: Classe que lida com escrita de arquivos.

MainClass
: Class principal que lê arquivos, calcula métricas e escreve o resultado em arquivos de saída.

4.1. Principais métodos implementados

Essa subseção irá listar os principais métodos implementados no sistema. Funções triviais não serão listadas, como *getters*, *setters* ou similares. Os métodos detalhados a seguir são os principais utilizados para calcular as métricas solicitadas pelo trabalho prático.

4.1.1. Pesquisador

- *public double calculaPesoImportancia()*

  Descrição: Calcula peso de importância do pesquisador.
  Retorno: Retorna o resultado no formato *double*.

- *public int totalCitacoes()*

  Descrição: Calcula o número de vezes que os artigos do pesquisador foram citados.
  Retorno: Retorna o resultado no formato *int*.

- *public double calculaPopularidade()*

  Descrição: Calcula popularidade do pesquisador, utiliza *calculaPesoImportancia()* e *totalCitacoes()* para realizar o cálculo. É sobrescrita nas classes filhas pois o cálculo irá se especializar.
  Retorno: Retorna o resultado no formato *double*.

4.1.2. Veículo Publicação

- *public int calculaCitacoes()*

  Descrição: Calcula o somatório do numero de citações de cada artigo presente em cada veiculo de publicação.
  Retorno: Retorna o resultado no formato *int*.

- *public double calculaFatorImpacto()*

  Descrição: Calcula o fator de impacto do veiculo de publicação, utiliza o *calcularCitacoes() para realizar o cálculo.*. Função é sobrescrita nas classes filhas para especializar o cálculo
  Retorno: Retorna o resultado no formato *double*.

4.1.3. Artigo

- *public double calculaPontuacaoArtigo(ArrayList <VeiculoPublicacao>listaVeiculosPublicacao)*

  Descrição: Calcula a pontuação do artigo com base no veículo de publicação e a quantidade de citações do artigo.
  Parâmetros: Utiliza uma lista com todos os veículos de publicação para encontrar aquele no qual o artigo está associado.
  Retorno: Retorna o resultado no formato *double*.

5. Testes

Realizamos testes utilizando as entradas fornecidas junto com o trabalho. Além disso criamos uma entrada menor para poder anexar junto à esse documento. Essa saída tem como objetivo apenas ilustrar o funcionamento do programa para uma entrada menor do que a fornecida inicialmente.

5.1. Listagem da saída

```
fatorImpacto_veiculo.txt
1;1.5000
2;1.5000
3;1.8750
```

4;1.5833

pontuacao_artigo.txt
1;1.5833
2;1.5000
3;0.0000
4;1.8750
5;1.8750
6;1.5000
7;3.7500
8;1.5000
9;3.1667
10;0.0000
11;0.0000
12;0.0000
13;1.5833
14;0.0000
15;1.8750
16;0.0000
17;0.0000
18;1.8750
19;0.0000
20;0.0000
21;1.8750
22;0.0000
23;1.5000
24;1.5833
25;0.0000
26;1.8750
27;0.0000
28;1.5000
29;1.5000
30;1.8750
31;0.0000
32;1.8750
33;0.0000
34;1.8750
35;0.0000
36;0.0000
37;0.0000
38;0.0000
39;0.0000
40;0.0000
41;0.0000
42;3.0000
43;1.5833

```
44;3.7500
45;1.5000
46;1.8750
47;1.5000
48;0.0000
49;1.5000
50;1.5833

popularidade_pesquisador.txt
1;917.7500
2;232.0000
3;325.8333
4;285.0000
5;418.0000
6;317.3333
7;496.3333
8;531.0000
9;320.5000
10;298.3333
11;361.0000
12;375.0000
13;520.0000
14;283.0000
15;314.5000
```

6. Conclusão

O trabalho prático foi prazeroso de ser realizado. Os conceitos de modelagem de classe aprendidos em sala foram bastante reforçados quando nos deparamos com um problema complexo como este.

A maior crítica em relação ao trabalho prático é em relação à mudança de especificações depois do trabalho ter sido lançado. Como ocorreu no nosso grupo, não tínhamos acessado o fórum de discussões até o momento em que nos deparamos com alguma dúvidas em relação à saída. Quando acessamos o fórum vimos que a saída esperada tinha mudado e novos arquivos estavam lá [UFMGVirtual].

No fórum também notamos que algumas especificações tinham mudado, como, por exemplo, a passagem dos arquivos de entrada (e saída) como parâmetro para o programa. Nossa sugestão é que caso ocorram mudanças na especificação do trabalho elas devem ser notificadas para todos os alunos de uma outra maneira. Caso contrário precisaremos ler todos os tópicos do fórum com uma certa regularidade para verificar se alguma especificação foi alterada.

Referências

UFMGVirtual. Forum do moodle. https://virtual.ufmg.br/20151/mod/forum/discuss.php?d=13081.

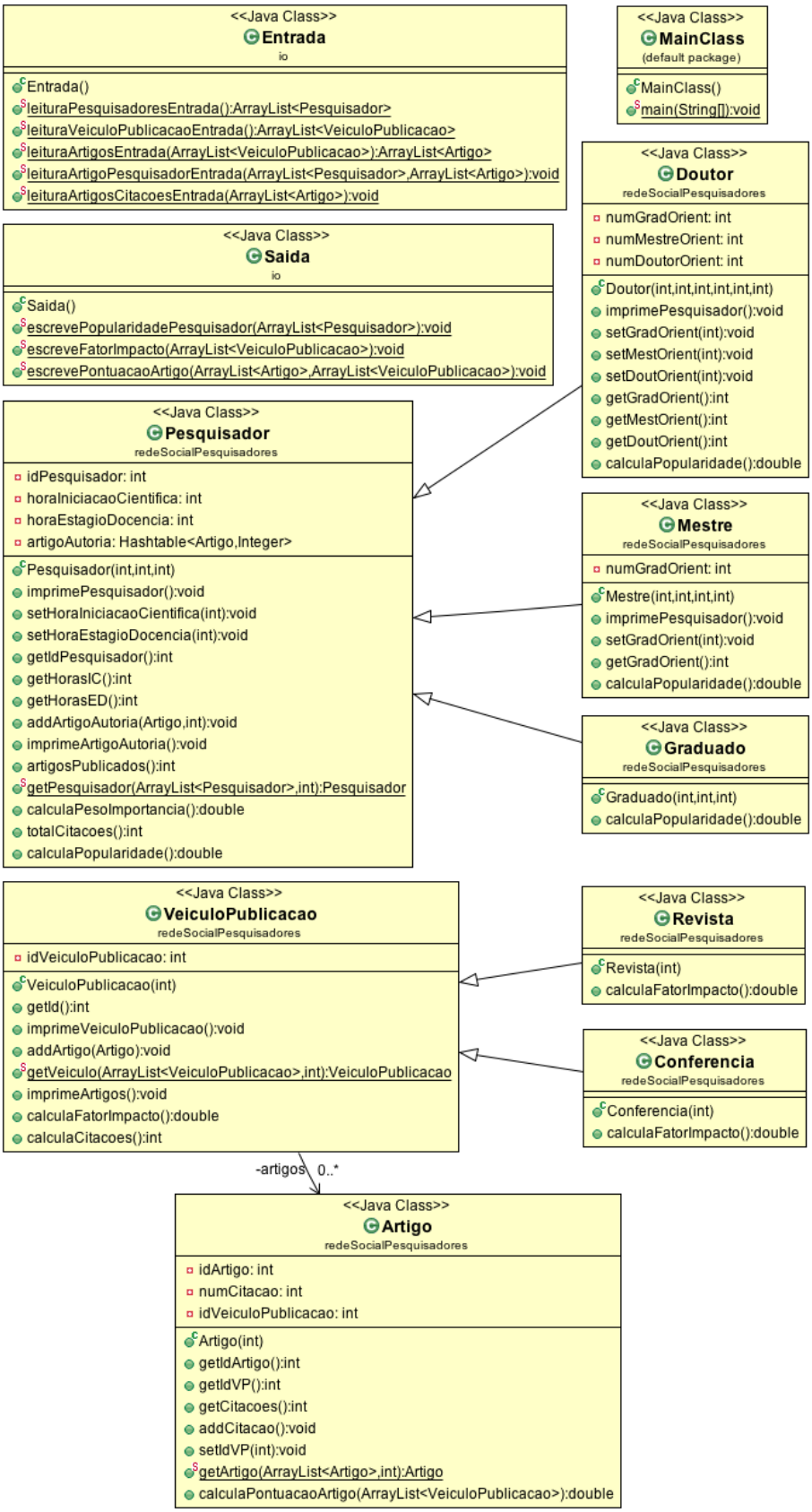

Figura 2. Diagrama de classes completo