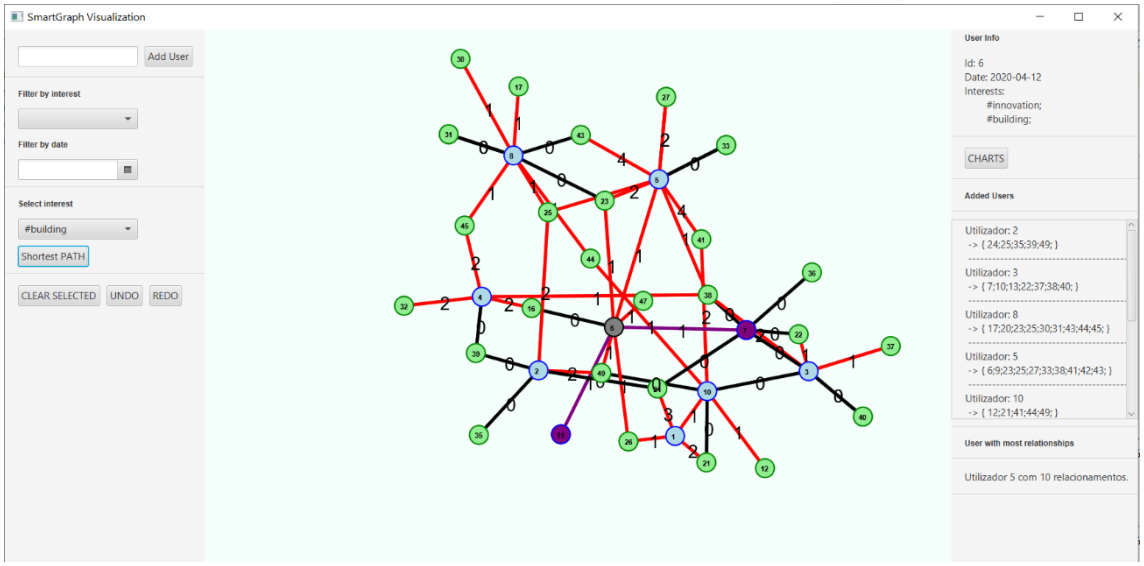
IPS Instituto Politécnico de Setúbal
Escola Superior de Tecnologia de Setúbal

Social Network



Programação Avançada

Ano letivo: 2020/2021

Docentes: Bruno Silva | André Sabino | Patrícia Macedo

Realizado por:

Bruno Carvalho – 160221028

João Madeira - 180221101

Índice

Apresentação dos TAD implementados

O projeto de recurso lecionado em Programação Avançada teve como objetivo utilizar os Tipos Abstratos de Dados aprendidos ao longo do ano letivo.

Para este trabalho foi proposto utilizarmos uma Lista de Adjacência ou uma Matriz de Adjacência.

Foi escolhido pela equipa de desenvolvimento a utilização da Lista de Adjacência por já ter sido utilizada em outros projetos e termos a experiência necessária para podermos manipular os seus dados.

Os Tipos Abstratos de Dados que foram utilizados neste projeto foram:

Graph – É um TAD que tem como função implementar grafos não direcionais ou unidirecionais. Consiste em guardar vértices ou nós e arestas que os ligam.

A Social Network tem como base uma classe designada por *GraphAdjacencyList.java* que estende os métodos da interface *Graph.java.* Isto significa que esta classe que foi estendida é uma variante do que se pode fazer com os métodos declarados na interface.

Para ser possível manipular e utilizar os tipos implementados em *Graph.java* é necessário termos duas classes privadas que estão alocadas em *GraphAdjacencyList.java.* Elas são MyEdge e *MyVertex.*

```
private class MyVertex implements Vertex<V> {

    V element;
    List<Edge<E, V>> edges;
```

MyVertex implementa a interface *Vertex<V>* e contém um elemento do tipo V e uma lista de arestas deste elemento.

```
private class MyEdge implements Edge<E, V> {

    E element;
    Vertex<V> vertexOutbound;
    Vertex<V> vertexInbound;
```

Esta classe privada implementa a interface *Edge<E, V>* e contém o elemento das arestas e vai complementar a lista de arestas definidas em *MyVertex*.

Map – É um TAD que se constrói através de uma propriedade *<Key, Value>* em que a *key* é um número inteiro e único e que o seu objetivo é guardar um ou mais valor.

Para a rede social em questão poder ser implementada de uma forma realista esta lista de adjacência (implementada em *GraphAdjacencyList.java*) é composta por uma *Key* do tipo V e os seus *Value* são do tipo Vertex<V>.

```
private final Map<V, Vertex<V>> graph;
```

V – É um elemento do tipo Vertex.

Vertex<V> - É um vértice que contém um elemento do tipo V.

Classes

Para a descrição das classes foram usados os comentários gerados no javadoc.

GraphAdjacencyList.java - This class implements the methods of the interface graph. It serves to make the social network happens. In this case, it was used an adjacency list.

Command.java - Interface of design pattern command.

CommandFilterDate.java - This classes makes the action when a client wants to filter the Social Network by a date.

CommandFilterInterest.java - This classes makes the action when a client wants to filter the Social Network by an interest.

CommandManager.java - Command Manager is a class that will have a stack and will give the possibility to make undo and redo of some client actions.

CommandRedo.java - This class will execute a possibility to many action of the client can be re-written.

CommandUndo.java - This class will execute a possibility to many action of the client can be restaured.

CommandUser.java - Command User is a class that will make the action of users automatic insertion after a undo or a redo.

CommandUserBatch.java - Command User Batch is a class that will make the action of users batch insertion after a undo or a redo.

Logger.java - Logger class is made to create a log.txt. log.txt is a file that will have some insertions and datestamps about users added by the client and their relationships.

LoggerProperties.java - Class that generate some properties that are boolean, wich mean that can be false or true and them are implemented to can be show if true or not if false in the log.txt file.

LoggerException.java - LoggerException is a class that if a exception is activated in a method that receives it, return a string.

Interest.java - Interest class is a class that defines the interest of an user.

InterestException.java - InterestException is a class that if a exception is activated in a method that receives it, return a string.

SocialNetwork.java - SocialNetwork class is a class that will have crucial methods that will make the possibility to construct the graph implemented in a specific Social Network and translate that to an User Interface more realistic.

SocialNetworkException.java - SocialNetworkException is a class that if a exception is activated in a method that receives it, return a string.

Relationship.java - Relationship class represents the connection of two users and that connection can be simple or shared interests accordingly to the type of that relation and their interest in common. The parameters of relationships are: User1 and User2, date, type of relationship mentioned over and the cost.

User.java - User class represents the parameters that an User has. {Type, number and creation date}.

ChartsTemplate.java - This ChartTemplate has the functionality to set a stage that will be used to make the 3 differentes types of Charts, only the variables will be not the same. Design Pattern - Template Method.

Top6Chart.java - Top6Chart represents the number of relationships with 6 users with most interests.

Top10ChartWithoutSharedInterests.java - Top10ChartWithSharedInterests represents the number of relationships with 10 users with most relationships without shared interests.

Top10ChartWithSharedInterests.java - Top10ChartWithoutSharedInterests represents the number of relationships with 10 users with most relationships with most shared interests.

SocialNetworkUI.java - SocialNetworkUI is a class that has the methods to transform the SocialNetwork in a User Interface that will help the the client to use it and the goal is to make the entire project more realistic and fucntional.

Padrões de Desenho

De acordo com o que foi lecionado durante o ano letivo era necessário aplicar os padrões de desenho no projeto. Para tal, foram implementados 5 padrões.

Command Pattern - Tem como objetivo encapsular uma solicitação como um objeto, o que lhe permite parametrizar outros objetos com diferentes solicitações, enfileirar solicitações e implementar recursos de cancelamento de operações. Isso inclui informações como o nome do método, o objeto que o método pertence e os valores dos parâmetros do método.

Este comando é utilizado na criação do grafo (em modo automático e em modo batch); É também utilizado para ser possível o utilizador poder voltar atrás (undo) ou para conseguir voltar a fazer algo em que tenha feito undo (Redo) e nos filtros, tanto para ver utilizadores pelo interesse como pela data.

Template Method – Tem como objetivo o auxílio na definição de um algoritmo com partes do mesmo definidos por métodos abstratos. As subclasses devem responsabilizar-se por estas partes abstratas, deste algoritmo, que serão implementadas, possivelmente de várias formas, ou seja, cada subclasse irá implementar à sua necessidade e oferecer um comportamento concreto construindo todo o algoritmo.

Este método é utilizado na criação das estatísticas para os gráficos. Tenho uma classe abstrata com 2 métodos abstratos, isto é, serão ambos utilizados nas diferentes subclasses mas o que varia é o algoritmo.

Model View Controller - Divide a aplicação em camadas interconectadas. Isto é feito para separar representações de informação internas da forma de como a informação é apresentada e processado levando ao desenvolvimento paralelo de maneira eficiente.

O modelo (Model) é composto pelas classes que vão servir a construção do modelo: SocialNetwork.java, User.java, Interest.java, Relationship.java e as suas exceções.

Controller: É composto pelo padrão Command que vai servir para manipulação da informação do modelo para a parte gráfica, isto é, visual.

View: O SocialNetworkUI.java é a classe onde se vai tratar da parte gráfica, em que, o utilizador vai ter acesso e que retrata graficamente o problema proposto.

Logger - É utilizado para gerar um ficheiro txt designado log.txt onde se pode observar a um *datestamp* da inserçãos dos utilizadores no grafo, bem como as suas relações e interesses. É possível também, através de uma classe de propriedades, poder customizar esse log.txt com a informação que deverá aparecer ou não

Observer - Define uma dependência *um-para-muitos* entre objetos de modo que quando um objeto muda o estado, todos seus dependentes são notificados e atualizados automaticamente. Permite que objetos interessados sejam avisados da mudança de estado ou outros eventos que ocorrem num outro objeto.

Utilizado no padrão MVC onde o SocialNetworkUI.java tem os seus updates que vão ser notificados no SocialNetwork.java.

Refactoring

Bad-Smells	Vezes detetadas	Técnica de Refactoring	Antes	Depois
Dead code	6	Apagar o código.	`public int getNumberOfU-sers() { return sn.numVerti-ces(); }`	-
Type Embedded name	1	Rename method.	Users()	getUsers()
Uncommunicative Name	2	Rename method.	setSelected() getSelected()	setSelectedVertex() getSelectedVertex()

Anexo: Diagrama de classes geral