

**Joanã Costa 180 221 109**

**Daniel Cordeiro 190 221 101**

**PROGRAMAÇÃO AVANÇADA**

**WebCrawler**



Índice

[Índice de Figuras 3](#_Toc30644067)

[1. Introdução 4](#_Toc30644068)

[2. Tipos Abstratos de Dados 4](#_Toc30644069)

[3. Diagrama de Classes 5](#_Toc30644070)

[4. Documentação de Classes 7](#_Toc30644071)

[CareTaker 7](#_Toc30644072)

[Factories 7](#_Toc30644073)

[HomeController 7](#_Toc30644074)

[Home View 7](#_Toc30644075)

[Home View.StopCriteria 7](#_Toc30644076)

[Link 8](#_Toc30644077)

[LoggerWriter 8](#_Toc30644078)

[Main 8](#_Toc30644079)

[MyDigraph 8](#_Toc30644080)

[MyEdge 8](#_Toc30644081)

[MyVertex 8](#_Toc30644082)

[SearchDepth 8](#_Toc30644083)

[SearchExpandedPages 8](#_Toc30644084)

[SearchIterative 8](#_Toc30644085)

[SearchPages 9](#_Toc30644086)

[WebCrawler 9](#_Toc30644087)

[WebCrawlerJson 9](#_Toc30644088)

[WebCrawlerFile 9](#_Toc30644089)

[WebPage 9](#_Toc30644090)

[5. Padrões de Software 9](#_Toc30644091)

[MVC e Observer 9](#_Toc30644092)

[DAO (Data Access Object) 10](#_Toc30644093)

[Strategy 10](#_Toc30644094)

[Simple Factory 11](#_Toc30644095)

[Singleton 11](#_Toc30644096)

[Memento 12](#_Toc30644097)

[6. Refactoring 12](#_Toc30644098)

[7. Conclusão 17](#_Toc30644099)

# Índice de Figuras

[Figura 1 - Diagrama de Classes 7](file:///D:\Engenharia\Ano%203\PA\PROJETO\Relatório.docx#_Toc30610151)

[Figura 2 - MVC 10](file:///D:\Engenharia\Ano%203\PA\PROJETO\Relatório.docx#_Toc30610152)

[Figura 3 - Classes dos tipos de ficheiros 10](file:///D:\Engenharia\Ano%203\PA\PROJETO\Relatório.docx#_Toc30610153)

[Figura 4 - Representação das Estratégias 11](#_Toc30610154)

[Figura 5 - Factory 11](#_Toc30610155)

[Figura 6 - Classe Singlenton 11](#_Toc30610156)

[Figura 7 - Implementação Memento 12](#_Toc30610157)

[Figura 8 - Implementação da inner class no modelo 12](#_Toc30610158)

# Introdução

Este projeto tem como objetivo a utilização do *TAD* (Tipos Abstratos Dados) ***Graph****,* ***Digraph*** e ***Queue*** na resolução do problema de percorrer páginas *web* e extrair as páginas associadas. Sendo assim foi desenvolvido um ***WebCrawler***que fará essa tarefa através dos algoritmos ***breadth-first*** e eventualmente o algoritmo ***Dijkstra***.

O objetivo principal do ***WebCrawler*** é receber um link de uma página inicial (***Root***), percorrer todos os links incidentes a este mesmo e recursivamente fazer o mesmo para as páginas visitadas nesses mesmos links. Este percurso efetuado irá ter de respeitar diferentes critérios. Estes critérios serão propostos pelo utilizador via interface construída em ***JavaFX***, sendo estes, critério de largura, profundidade, modo iterativo e também o máximo de “**outbound’s**” links que uma página pode ter.

Á medida que as **páginas** são inseridas como ***vertex’s*** no ***Graph*** ligadas por **links**, que serão os ***edges***, é atualizado algumas estatísticas pertinentes (nº de páginas visitadas, páginas não encontradas, nº protocolos HTTP e nº de links existentes no grafo) e é também atualizado o **graph view** do ***JavaFX*** utilizando a biblioteca fornecida pelo *Professor Bruno Silva* no seguinte link do GitHub:

<https://github.com/brunomnsilva/JavaFXSmartGraph>

Com a interface gráfica e utilizando os métodos fornecidos pela biblioteca do Professor é dado a possibilidade de manipular, via *lambda functions*, a interação do utilizador para com o grafo existente no graph view, dando a possibilidade de abrir estas mesmas páginas no browser após um duplo clique num vértice, percorrer o grafo em modo iterativo e também mexer o grafo(funcionalidade default do grafo).

# Tipos Abstratos de Dados

Para este projeto utilizamos alguns tipos de dados para facilitar a manipulação dos dados. Estes são:

* ***HashMap*** – Usado para guardar os vértices e as arestas na classe ***MyDigraph***;
* ***ArrayList*** – Usado para guardar vários tipos de dados ao longo do código;
* ***Queue*** – Usado para armazenar em memória as páginas Web para que sejam retiradas por ordem e processadas;

Tipos abstratos de dados implementados:

* ***MyDigraph*** – Tipo de dados que implementa o tipo ***Digraph*** que representa o grafo que irá ser utilizado com as páginas web e links das mesmas.
* ***MyVertex*** – Tipo de dados que representa um vértice do grafo (será cada **WebPage**);
* ***MyEdge*** – Tipo de dados que representa uma aresta do grafo(será cada **Link**);

# Diagrama de Classes

Na figura que se segue estão representadas as classes utilizadas na 1º fase do desenvolvimento da aplicação do ***WebCrawler*** e os acoplamentos entre elas.

Uma imagem com texto

Descrição gerada automaticamente

Uma imagem com captura de ecrã, texto

Descrição gerada automaticamente

Uma imagem com captura de ecrã

Descrição gerada automaticamente

Uma imagem com captura de ecrã

Descrição gerada automaticamente

Figura 1 - Diagrama de Classes

# Documentação de Classes

## CareTaker

Classe que gere as solicitações para guardar e restaurar os estados do objeto (**WebCrawler**).

## Factories

Classe responsável por criar a estrutura *MVC* (**Model-View-Controller**) e os tipos de arquivo a serem exportados (“**DATA**” e “**JSON”**).

## HomeController

Classe responsável por gerenciar toda a lógica de negócio entre o modelo (**WebCrawler**) e a vista (**HomeView**).

## Home View

Classe que contém toda a implementação sobre a interface do utilizador. Implementa um comportamento da visualização e um observador para manter-se atualizado quando o modelo for alterado.

## Home View.StopCriteria

Enumerado que identifica os critérios de pesquisa que manipulam a construção do **WebCrawler**. Os valores são: **PAGES**, **DEPTH**, **ITERATIVE** e **EXPANDED**;

## Link

Classe que representa um link de uma página web. Este representará o tipo genérico que será guardado no grafo.

## LoggerWriter

Classe responsável por gravar mensagens em um ficheiro de texto

## Main

Classe que é ponto principal de entrada do programa.

## MyDigraph

Classe que implementa um grafo. Para além dos métodos normais de criação e manipulação de um grafo existem também métodos para retornar vértices aleatórios.

## MyEdge

Classe privada criada na classe **MyDigraph** que implementa uma aresta (**Edge**) do grafo. Contém um elemento e o vértice de início e fim.

## MyVertex

Classe privada criada na classe **MyDigraph** que implementa um vértice (**Vertex**) do grafo. Contém um elemento.

## SearchDepth

Classe que implementa de **ISearchCriteria** e é uma estratégia para pesquisar páginas por modo automático dado um critério profundidade. Neste modo a construção do grafo termina assim que todos as páginas que “distam” M links da página inicial forem visitadas.

## SearchExpandedPages

Classe que implementa de **ISearchCriteria** e é uma estratégia para pesquisar páginas por modo automático dado um critério de máximo de links de uma página visitada. Neste modo na construção do grafo as páginas que tem mais do que n links, não serão mais expandidas.

## SearchIterative

Classe que implementa de **ISearchCriteria** e é uma estratégia para pesquisar páginas por modo iterativo. Neste modo é o utilizador que determina o ritmo e ordem de visita das páginas e, subsequentemente, da geração do modelo e sua visualização.

## SearchPages

Classe que implementa de **ISearchCriteria** e é uma estratégia para pesquisar páginas por modo automático dado um critério de máximo de páginas visitadas. Neste modo a construção do grafo é feita em ***breadth-first*** e termina assim que N páginas forem visitadas;

## WebCrawler

Esta classe é responsável por criar o nosso modelo do **WebCrawler** baseando-se na classe **MyDigraph**, onde os vértices são representados pelas páginas web (**WebPage**) e arestas por um link desta página (classe Link). Esta classe inclui também grande parte da manipulação das páginas, obtenção de estatísticas e contagens efetuadas, verificações acerca das paginas, *restart* do grafo utilizado e output retornado na consola com a informação pertinente da sua utilização.

## WebCrawlerJson

Classe que gera um novo ficheiro do tipo ***JSON*** pela serialização do java. Implementa a interface **IWebCrawlerDAO** que faz operações para guardar e carregar o ficheiro pretendido.

## WebCrawlerFile

Classe gera um novo ficheiro de texto pela serialização do java. Implementa a interface **IWebCrawlerDAO** que faz operações para guardar e carregar o ficheiro pretendido.

## WebPage

Classe que representa o vértice (**Vertex**) no gráfico. Esta classe é representação de uma página web em concreto. Contém os atributos pertinentes sobre a página web (url, status code, outbound links, titulo da pagina).

# Padrões de Software

Durante o decorrer da unidade curricular foram lecionados alguns padrões de software dos quais os utilizados foram:

## MVC e Observer

**MVC** é o acrônimo de **Model-View-Controller**. Este padrão separa-se em 3 camadas essenciais, Camada do modelo ou lógica da aplicação (**Model**), camada da apresentação (**View**), camada do controlador (**Controller**).

A estrutura do nosso projeto é baseada neste padrão devido a sua robustez e distribuição de responsabilidade entre classes, cada classe é responsável pelo que lhe foi imputado. Sendo assim temos três packages no nosso projeto, **Model, Views e Controller**. Na figura abaixo temos representado a estrutura no ***NetBeans***.

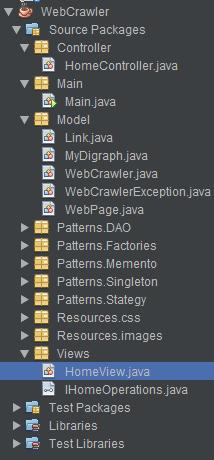


Figura 2 - MVC

## DAO (Data Access Object)

Como no nosso projeto seria preciso exportar ficheiros de texto e em JSON, tivemos que implementar este padrão para esses dois tipos de ficheiro. Este está representado na classe ***WebCrawlerFile*** e ***WebCrawlerJson*** onde é usado o modelo (***WebCrawler***).



Figura 3 - Classes dos tipos de ficheiros

## Strategy

O padrão **Strategy** é um padrão de design comportamental que permite definir uma família de algoritmos (presentes na classe **ISearchCriteria**), colocar cada um deles em uma classe separada e tornar seus objetos intercambiáveis.

Este padrão foi utilizado para alternar entre os critérios de paragem no modo iterativo e nos modos automáticos.

No modo automático temos três critérios de paragem sendo eles:

* **SearchPages -** Termina assim que n páginas forem visitadas;
* **SearchDepth** - Termina assim que todas as páginas de que distam M links da página inicial;
* **SearchExpandedPages** - Uma página que tem mais do que n links, não será mais expandida;
* **SearchIterative** – As páginas são visitadas em modo iterativo consoante a interação do utilizador

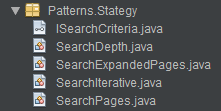


Figura 4 - Representação das Estratégias

## Simple Factory

Este padrão usámos para criar objetos pertencentes ao padrão MVC e os tipos de ficheiro juntamente com o DAO. As classes usadas são a ***Main, WebCrawler e WebPage***.



Figura 5 – Factory

## Singleton

É um padrão de design de criação que permite garantir que uma classe tenha apenas **uma instância**, enquanto fornece um ponto de **acesso global** a essa mesma instância apartir do metódo **getInstance**.

Usámos este padrão para construir e dar acesso aos ***logs*** por ficheiro de texto do programa. Este é chamado em quase todos as classes do programa para notificar **erros** que ocorram durante a execução do programa e para escrever as páginas web que foram inseridos durante a pesquisa das mesmas.

Classes Usadas:

* WebCrawler
* WebPage
* WebCrawlerFile
* WebCrawlerJson
* SearchPages
* SearchIterative
* SearchDepth

Figura 6 - Classe Singlenton

## Memento

É um padrão de design comportamental que permite **gravar** e **restaurar** o **estado** anterior de um objeto sem revelar os detalhes de sua implementação.

Apesar de não termos conseguido implementar de forma perfeita no nosso projeto, este também está presente na classe principal de modelo do ***WebCrawler***, onde o objetivo era ir guardando os estados do grafo quando a página era expandida e permitir fazer o **undo** dessa ação se o utilizador quisesse.

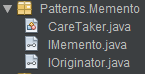


Figura 7 - Implementação Memento

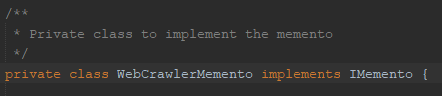


Figura 8 - Implementação da inner class no modelo

# Refactoring

Nesta parte, vamos começar a fazer toda busca por código que não esteja bem ao longo das classes do projeto (***bad smells***) por isso usamos técnicas que irão melhorar o aspeto do código fazendo com que outro programador ao ver o código seja mais fácil de ler o código dai chamado o ***Refactoring***.

|  |  |
| --- | --- |
| Bad Smell | Duplicated Code |
| Técnica de Refactoring | Extract Method |
| Código Antigo |  |
| Código Resultante |  |

|  |  |
| --- | --- |
| Bad Smell | Data Clump |
| Técnica de Refactoring | Extract Class + Hide delegate |
| Código Antigo |  |
| Código Resultante | Na classe WebCrawler:  Uma imagem com monitor, texto  Descrição gerada automaticamente |

|  |  |
| --- | --- |
| Bad Smell | Innappropriate Intimacy |
| Técnica de Refactoring | Hide Delegate |
| Código Antigo |  |
| Código Resultante |  |

|  |  |
| --- | --- |
| Bad Smell | Large Class |
| Técnica de Refactoring | Extract SubClass |
| Código Antigo |  |
| Código Resultante |  |

|  |  |
| --- | --- |
| Bad Smell | Makink Method Calls Simpler |
| Técnica de Refactoring | Rename method |
| Código Antigo |  |
| Código Resultante |  |

|  |  |
| --- | --- |
| Bad Smell | Message Chain |
| Técnica de Refactoring | Hide Delegate |
| Código Antigo | this.logW.webPageInsertWriteToLog(webPageInserting, visitedWebPage, this.model.getGraph().  incidentEdges(this.model.getEqualWebPageVertex(webPageInserting.getPersonalURL())).size()); |
| Código Resultante | this.logW.webPageInsertWriteToLog(webPageInserting, visitedWebPage, this.model.getIncidentLinksSize(webPageInserting)); |

# Conclusão

Neste projeto conseguimos absorver todos os conhecimentos lecionados durante este semestre, desde os **TAD’s**, **padrões de software** a **refactoring**.

**Rafael Trindade - 160210033**

**Vítor Gomes - 160210050**

Com importância mais dos padrões de software onde o principal ponto é detetar padrões a partir de problemas encontrados durante o desenvolvimento do programa e aplicar o padrão mais indicado.

E por fim também temos o refactoring outro ponto importante no desenvolvimento de software dado que melhora a leitura do código para outro programador e também por consequência deixa o software mais robusto, estruturado e eventualmente melhoria do desempenho do programa.