PROGRAMAÇÃO AVANÇADA

Joanã Costa 180 221 109
Daniel Cordeiro 190 221 101

WebCrawler

IPS Instituto Politécnico de Setúbal
Escola Superior de Tecnologia de Setúbal

Índice

Índice de Figuras

1. 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).

2. Tipos Abstratos de Dados

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

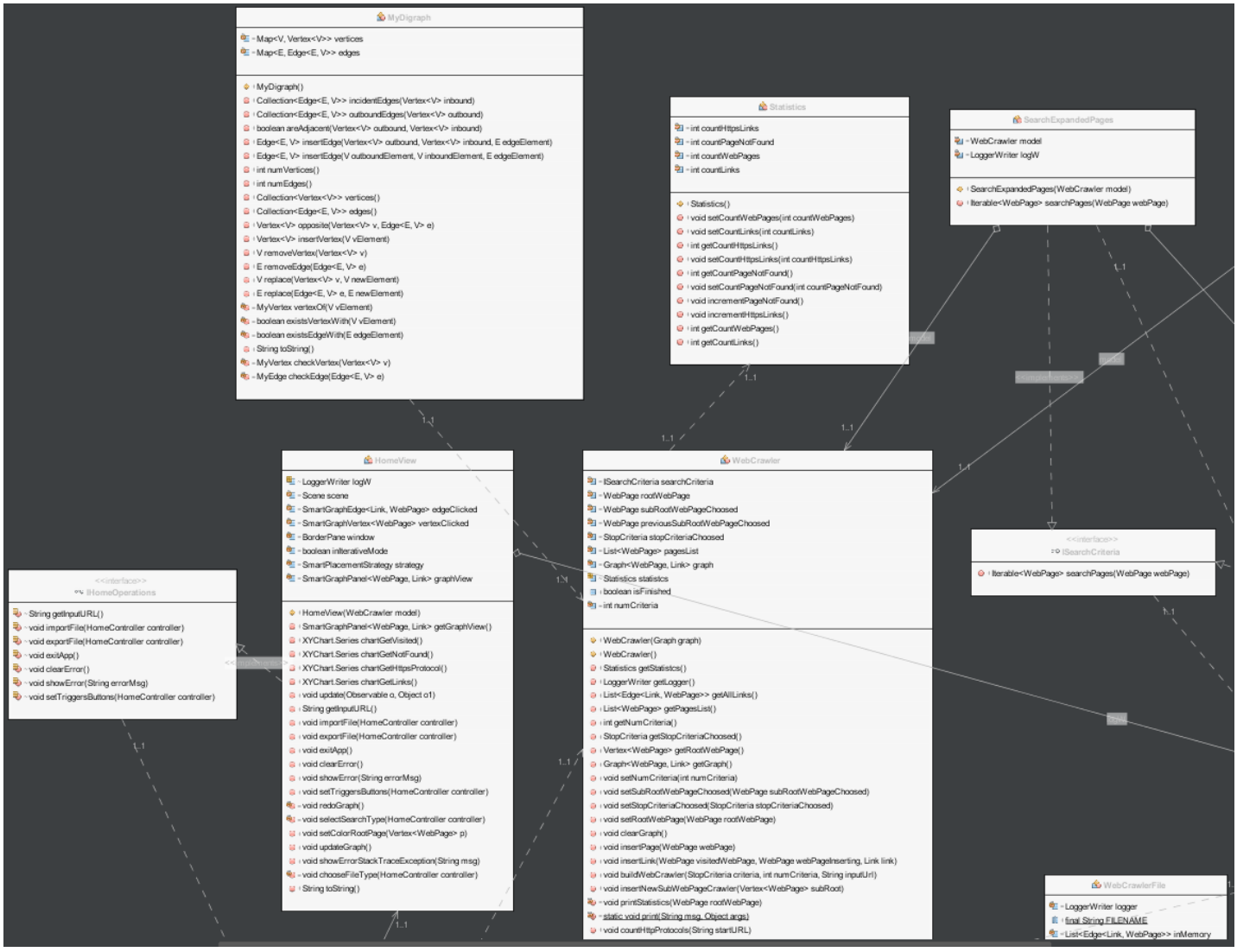
- ***HashMap*** – Usado para guardar os vértices e as arestas na classe ***MyDigraph***;
- ***ArrayList*** – Usado para guardar vários tipos de dados ao longo do código;
- ***Queue*** – Usado para armazenar em memória as páginas Web para que sejam retiradas por ordem e processadas;

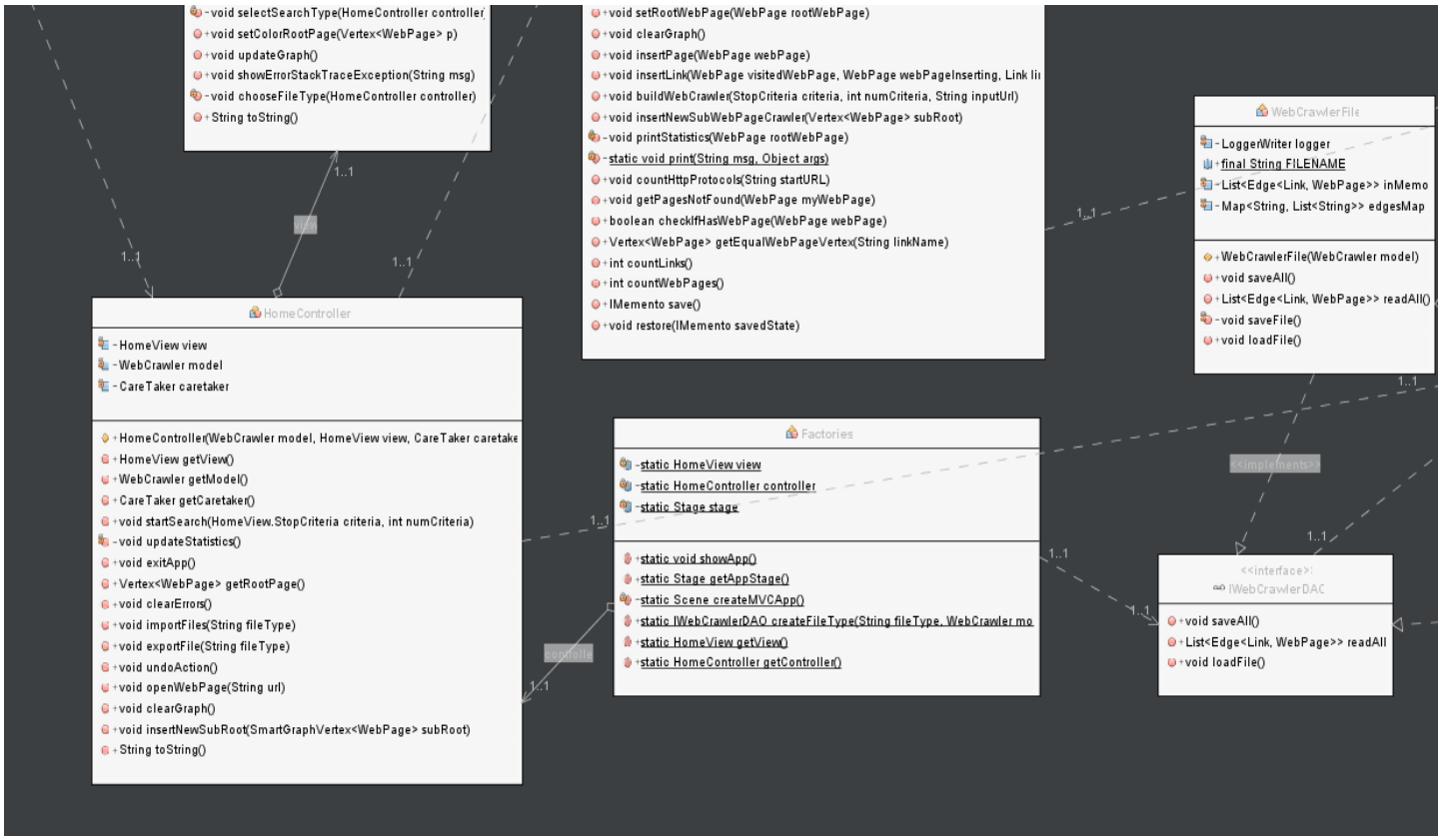
Tipos abstratos de dados implementados:

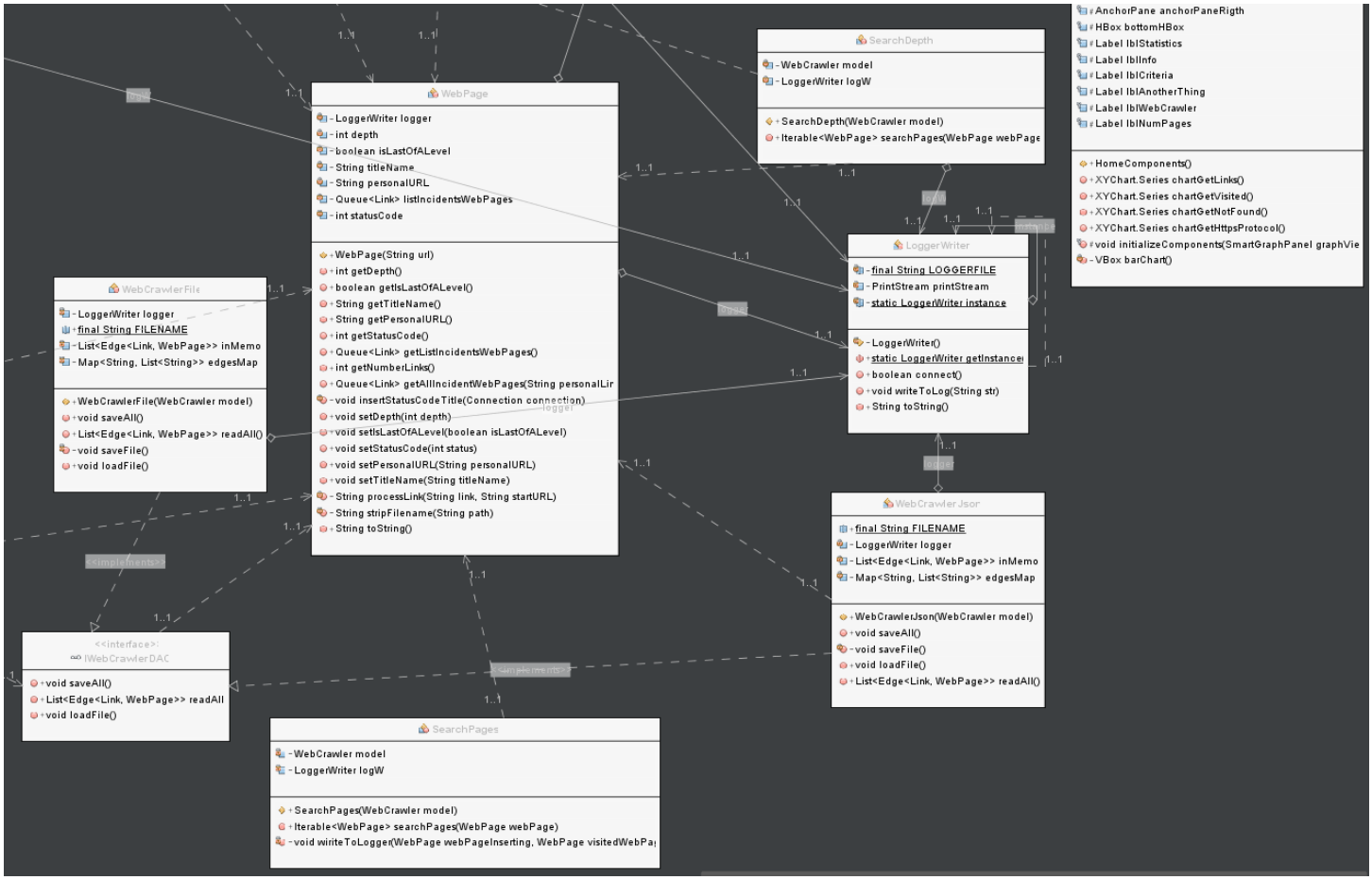
- ***MyDigraph*** – Tipo de dados que implementa o tipo ***Digraph*** que representa o grafo que irá ser utilizado com as páginas web e links das mesmas.
- ***MyVertex*** – Tipo de dados que representa um vértice do grafo (será cada **WebPage**);
- ***MyEdge*** – Tipo de dados que representa uma aresta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

3. 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.




-void selectSearchType(HomeController controller
+void setColorRootPage(Vertex<WebPage> p)
+void updateGraph()
+void showErrorStackTraceException(String msg)
-void chooseFileType(HomeController controller)
+String toString()
+void setRootWebPage(WebPage rootWebPage)
+void clearGraph()
+void insertPage(WebPage webPage)
+void insertLink(WebPage visitedWebPage, WebPage webPageInserting, Link li
+void buildWebCrawler(StopCriteria criteria, int numCriteria, String inputUrl)
+void insertNewSubWebPageCrawler(Vertex<WebPage> subRoot)
-void printStatistics(WebPage rootWebPage)
-static void print(String msg, Object args)
+void countHttpProtocols(String startURL)
+void getPagesNotFound(WebPage myWebPage)
+boolean checkIfHasWebPage(WebPage webPage)
+Vertex<WebPage> getEqualWebPageVertex(String linkName)
+int countLinks()
+int countWebPages()
+IMemento save()
+void restore(IMemento savedState)
WebCrawlerFile
-LoggerWriter logger
+final String FILENAME
-List<Edge<Link, WebPage>> inMemo
-Map<String, List<String>> edgesMap
+WebCrawlerFile(WebCrawler model)
+void saveAll()
+List<Edge<Link, WebPage>> readAll()
-void saveFile()
+void loadFile()
1..1
HomeController
-HomeView view
-WebCrawler model
-CareTaker caretaker
+HomeController(WebCrawler model, HomeView view, CareTaker caretake
+HomeView getView()
+WebCrawler getModel()
+CareTaker getCaretaker()
+void startSearch(HomeView.StopCriteria criteria, int numCriteria)
-void updateStatistics()
+void exitApp()
+Vertex<WebPage> getRootPage()
+void clearErrors()
+void importFiles(String fileType)
+void exportFile(String fileType)
+void undoAction()
+void openWebPage(String url)
+void clearGraph()
+void insertNewSubRoot(SmartGraphVertex<WebPage> subRoot)
+String toString()
Factories
-static HomeView view
-static HomeController controller
-static Stage stage
+static void showApp()
+static Stage getAppStage()
-static Scene createMVCApp()
+static IWebCrawlerDAO createFileType(String fileType, WebCrawler mo
+static HomeView getView()
+static HomeController getController()
<<interface>>
IWebCrawlerDAO
+void saveAll()
+List<Edge<Link, WebPage>> readAll
+void loadFile()
<<implements>>
controller
1..1


WebPage
-LoggerWriter logger
-int depth
-boolean isLastOfALevel
-String titleName
-String personalURL
-Queue<Link> listIncidentsWebPages
-int statusCode
+WebPage(String url)
+int getDepth()
+boolean getIsLastOfALevel()
+String getTitleName()
+String getPersonalURL()
+int getStatusCode()
+Queue<Link> getListIncidentsWebPages()
+int getNumberLinks()
+Queue<Link> getAllIncidentWebPages(String personalLir
-void insertStatusCodeTitle(Connection connection)
+void setDepth(int depth)
+void setIsLastOfALevel(boolean isLastOfALevel)
+void setStatusCode(int status)
+void setPersonalURL(String personalURL)
+void setTitleName(String titleName)
-String processLink(String link, String startURL)
-String stripFilename(String path)
+String toString()
SearchDepth
-WebCrawler model
-LoggerWriter logW
+SearchDepth(WebCrawler model)
+Iterable<WebPage> searchPages(WebPage webPage
LoggerWriter
-final String LOGGERFILE
-PrintStream printStream
-static LoggerWriter instance
-LoggerWriter()
+static LoggerWriter getInstance()
+boolean connect()
+void writeToLog(String str)
+String toString()
WebCrawlerFile
-LoggerWriter logger
+final String FILENAME
-List<Edge<Link, WebPage>> inMemo
-Map<String, List<String>> edgesMap
+WebCrawlerFile(WebCrawler model)
+void saveAll()
+List<Edge<Link, WebPage>> readAll()
-void saveFile()
+void loadFile()
WebCrawlerJson
+final String FILENAME
-LoggerWriter logger
-List<Edge<Link, WebPage>> inMemo
-Map<String, List<String>> edgesMap
+WebCrawlerJson(WebCrawler model)
+void saveAll()
-void saveFile()
+void loadFile()
+List<Edge<Link, WebPage>> readAll()
<<interface>>
IWebCrawlerDAO
+void saveAll()
+List<Edge<Link, WebPage>> readAll
+void loadFile()
<<implements>>
SearchPages
-WebCrawler model
-LoggerWriter logW
+SearchPages(WebCrawler model)
+Iterable<WebPage> searchPages(WebPage webPage)
-void wiriteToLogger(WebPage webPageInserting, WebPage visitedWebPa
#AnchorPane anchorPaneRigth
#HBox bottomHBox
#Label lblStatistics
#Label lblInfo
#Label lblCriteria
#Label lblAnotherThing
#Label lblWebCrawler
#Label lblNumPages
+HomeComponents()
+XYChart.Series chartGetLinks()
+XYChart.Series chartGetVisited()
+XYChart.Series chartGetNotFound()
+XYChart.Series chartGetHttpsProtocol()
#void initializeComponents(SmartGraphPanel graphVie
-VBox barChart()
1..1

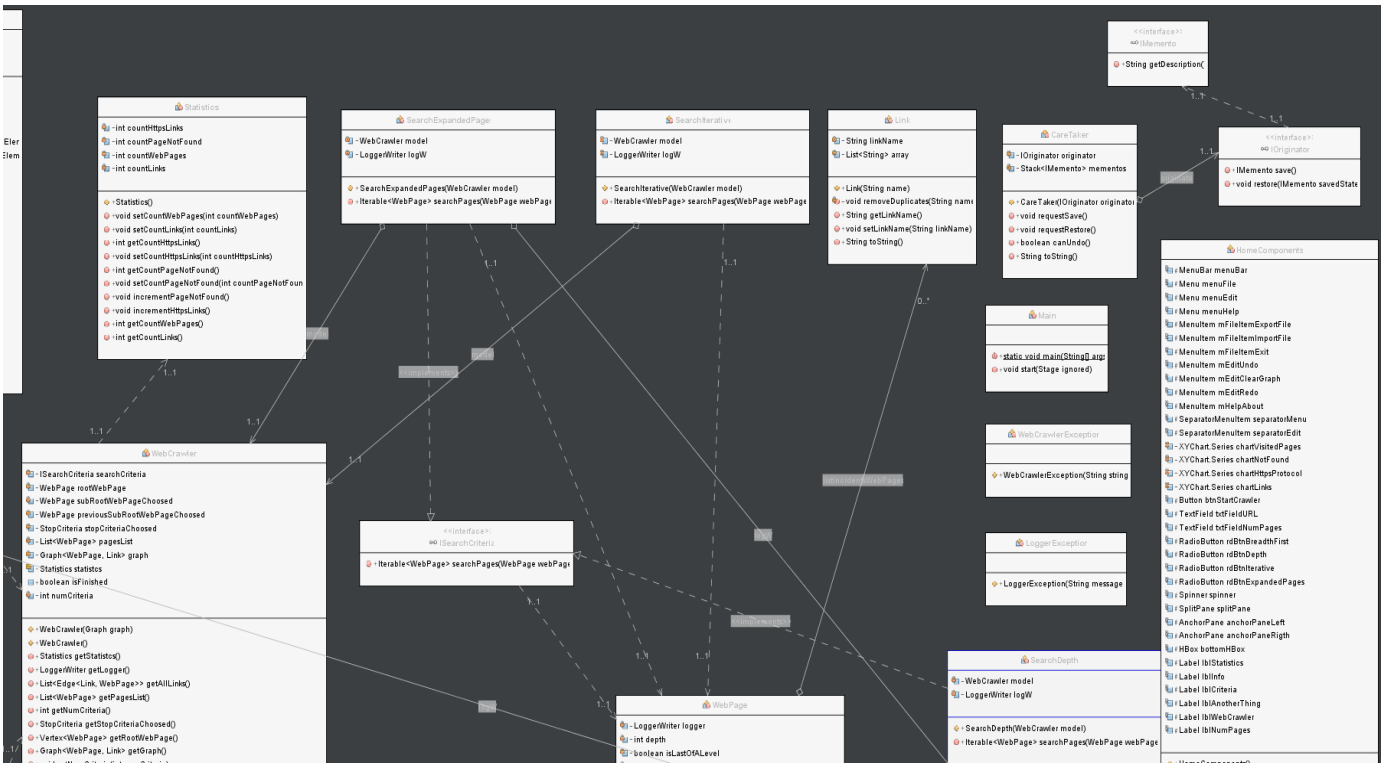


Figura 1 - Diagrama de Classes

4. 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).

5. 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***.

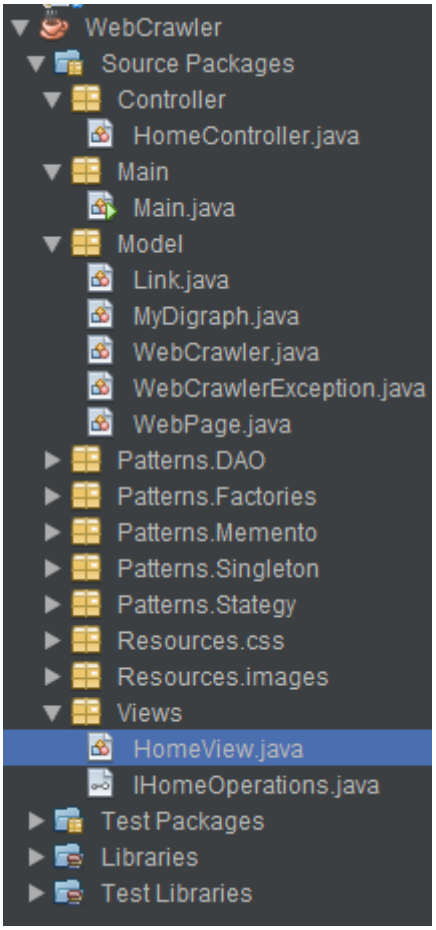


Figura 2 - MVC

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***).

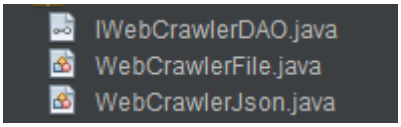


Figura 3 - Classes dos tipos de ficheiros

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

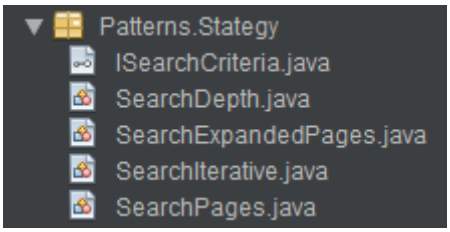


Figura 4 - Representação das Estratégias

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***.

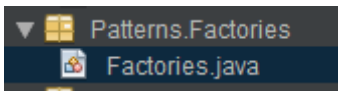


Figura 5 – Factory

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

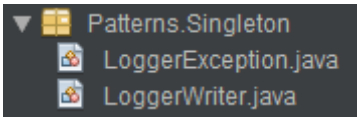


Figura 6 - Classe Singlenton

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.

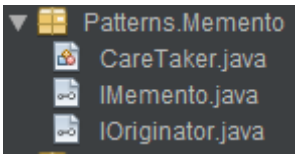


Figura 7 - Implementação Memento

```
/**
 * Private class to implement the memento
 */
private class WebCrawlerMemento implements IMemento {
```

Figura 8 - Implementação da inner class no modelo

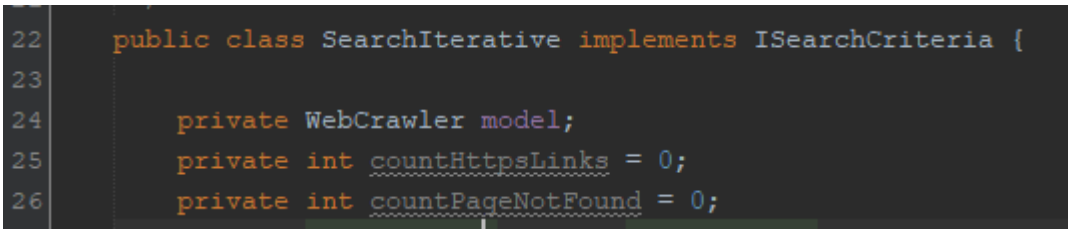
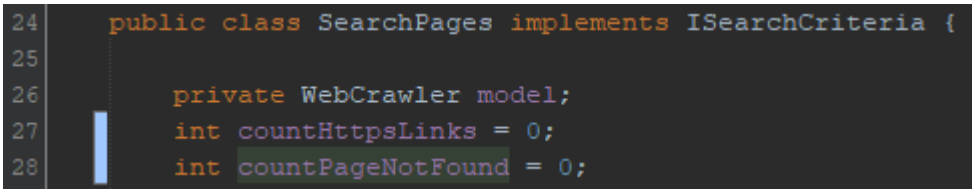
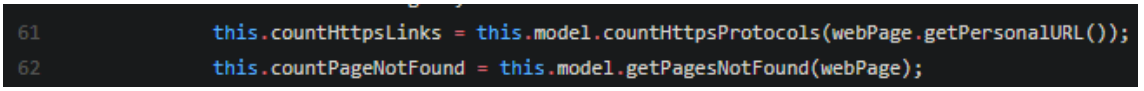
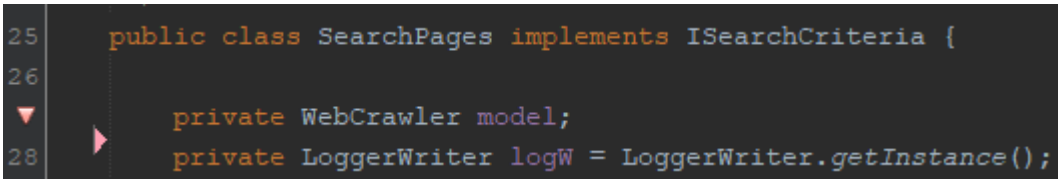
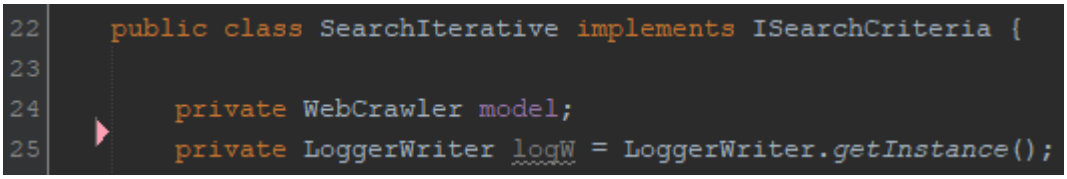
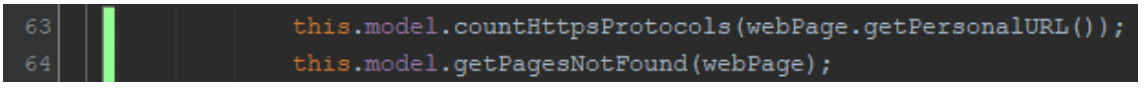
6. Refactoring

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	`countHttpsLinks += this.model.countHttpsProtocols(webPage.getPersonalURL());`
	`this.model.setCountHttpsLinks(countHttpsLinks);`
	`countPageNotFound += this.model.getPagesNotFound(webPage);`
	`this.model.setCountPageNotFound(countPageNotFound);`
	
	`countPageNotFound += this.model.getPagesNotFound(webPageInserting);`
	`countHttpsLinks += this.model.countHttpsProtocols(webPage.getPersonalURL());`
	`this.model.setCountHttpsLinks(countHttpsLinks);`
	`countPageNotFound += this.model.getPagesNotFound(webPage);`
	`this.model.setCountPageNotFound(countPageNotFound);`

Código Resultante	`updateStatisticsHttpsCount(webPage);`
	`updateStatisticsPageNotFound(webPage);`
	
	`updateStatisticsHttpsCount(visitedWebPage);`
	`updateStatisticsPageNotFound(visitedWebPage);`

Bad Smell	**Data Clump**
Técnica de Refactoring	Extract Class + Hide delegate
Código Antigo	  
Código Resultante	   Na classe WebCrawler:

```
22  public class SearchIterative implements ISearchCriteria {
23
24      private WebCrawler model;
25      private int countHttpsLinks = 0;
26      private int countPageNotFound = 0;
```

```
24  public class SearchPages implements ISearchCriteria {
25
26      private WebCrawler model;
27      int countHttpsLinks = 0;
28      int countPageNotFound = 0;
```

```
61          this.countHttpsLinks = this.model.countHttpsProtocols(webPage.getPersonalURL());
62          this.countPageNotFound = this.model.getPagesNotFound(webPage);
```

```
25  public class SearchPages implements ISearchCriteria {
26
27      private WebCrawler model;
28      private LoggerWriter logW = LoggerWriter.getInstance();
```

```
22  public class SearchIterative implements ISearchCriteria {
23
24      private WebCrawler model;
25      private LoggerWriter logW = LoggerWriter.getInstance();
```

```
63          this.model.countHttpsProtocols(webPage.getPersonalURL());
64          this.model.getPagesNotFound(webPage);
```

```
58      // Statistics
59      Statistics statistcs;
60      public boolean isFinished = false;
61      private int numCriteria = 0;
62
63  //</editor-fold>
64      public WebCrawler(Graph graph) {
65          this.graph = graph;
66          this.statistcs = new Statistics();
67      }
```

```
282     /**
283      * Count HTTPS protocols
284      *
285      * @param startURL site URL
286      * @throws MalformedURLException
287      */
288     public void countHttpsProtocols(String startURL) throws MalformedURLException {
289         URL u = new URL(startURL);
290         if (u.getProtocol().equals("http")) {
291             statistcs.incrementHttpsLinks();
292         }
293     }
```

Bad Smell	**Innappropriate Intimacy**
Técnica de Refactoring	Hide Delegate
Código Antigo	`// Insert the webPage in the graph` `this.model.getGraph().insertVertex(webPage);`
Código Resultante	`// Insert the webPage in the graph` `this.model.insertPage(webPage);`

Bad Smell	**Large Class**
Técnica de Refactoring	Extract SubClass
Código Antigo	

```java
public class HomeView extends VBox implements Observer, IHomeOperations {

    // Enum for searchCriteria
    public enum StopCriteria {
        PAGES, DEPTH, ITERATIVE;
    }

    LoggerWriter logW = LoggerWriter.getInstance();
    //SmartPlacementStrategy strategy = new SmartCircularSortedPlacementStrategy();
    //SmartPlacementStrategy strategy = new SmartRandomPlacementStrategy();
    private SmartPlacementStrategy strategy;
    private SmartGraphPanel<WebPage, Link> graphView;

    //Menu
    private MenuBar menuBar;
    private Menu menuFile;
    private Menu menuEdit;
    private Menu menuHelp;
    private MenuItem mFileItemExportFile;
    private MenuItem mFileItemImportFile;
    private MenuItem mFileItemExit;
    private MenuItem mEditUndo;
    private MenuItem mEditClearGraph;
    private MenuItem mEditRedo;
    private MenuItem mHelpAbout;
    private SeparatorMenuItem separatorMenu;
    private SeparatorMenuItem separatorEdit;

    //Actions left panel
    private Button btnStartCrawler;
    private TextField txtFieldURL;
    private TextField txtFieldNumPages;
    private RadioButton rdBtnBreadthFirst;
    private RadioButton rdBtnDepth;
    private RadioButton rdBtnIterative;
    private final Spinner spinner = new Spinner();

    //Layout
    private SplitPane splitPane;
    private AnchorPane anchorPaneLeft;
    private AnchorPane anchorPaneRigth;
    private HBox bottomHBox;

    //Labels
```

Código Resultante

```
70
71  /** This class contains all implementation about the UI ...9 lines */
80  public class HomeView extends HomeComponents implements Observer, IHomeOperations {
81
82      /** This ENUM represents the stop criteria type ...3 lines */
85      public enum StopCriteria {...3 lines }
88
89      LoggerWriter logW = LoggerWriter.getInstance();
90
91      private Scene scene;
92
93      // Graph interface
94      private SmartGraphEdge<Link, WebPage> edgeClicked = null;
95      private SmartGraphVertex<WebPage> vertexClicked;
96
97      private BorderPane window;
98      private boolean inIterativeMode = false;
99
100     private SmartPlacementStrategy strategy;
101     private SmartGraphPanel<WebPage, Link> graphView;
102
103     public HomeView(WebCrawler model) {...15 lines }
118
119     public SmartGraphPanel<WebPage, Link> getGraphView() {...3 lines }
```

Bad Smell	**Makink Method Calls Simpler**
Técnica de Refactoring	Rename method
Código Antigo	

```
266     * This method helps to see the pages that was generated.
267     *
268     * @param rootWebPage A object of the type WebPage
269     */
270    private void searchPagesAndPrint(WebPage rootWebPage) {
271        Iterable<WebPage> it = searchCriteria.searchPages(rootWebPage);
272
273        print("\n ========= Estatísticas ========= \n");
274        print(" »»»»» Páginas Visitadas (%d) «««««« \n\n %s", this.countWebPages(), it);
275        print(" »»»»» Páginas não encontradas (%d) ««««««", this.statistcs.getCountPageNotFound());
276        print(" »»»»» Ligações HTTPS (%d) ««««««", this.statistcs.getCountHttpsLinks());
277        print(" »»»»» Ligações entre páginas (%d) ««««««", this.countLinks());
```

Código Resultante

```
265    /**
266     * This method helps to see the pages that was generated.
267     *
268     * @param rootWebPage A object of the type WebPage
269     */
270    private void printStatistics(WebPage rootWebPage) {
271        Iterable<WebPage> it = searchCriteria.searchPages(rootWebPage);
272
273        print("\n ========= Estatísticas ========= \n");
274        print(" »»»»» Páginas Visitadas (%d) «««««« \n\n %s", this.countWebPages(), it);
275        print(" »»»»» Páginas não encontradas (%d) ««««««", this.statistcs.getCountPageNotFound());
276        print(" »»»»» Ligações HTTPS (%d) ««««««", this.statistcs.getCountHttpsLinks());
277        print(" »»»»» Ligações entre páginas (%d) ««««««", this.countLinks());
278
```

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));

7. Conclusão

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

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.