IPS Instituto Politécnico de Setúbal

Escola Superior de Tecnologia de Setúbal

Programação Avançada

2019/2020

Versão Final

Turma: SW-4

Docente: Bruno Silva

Alunos: Diogo Venâncio e Miguel Lapão

Número: 160221076 - 170221003

Índice

TADs implementadas 3
Queue 3
Stack 3
DiGraph 3
Diagrama de classes 4
Classes 4
projecto.model 4
projecto 5
com.brunomnsilva.smartgraph.containers 5
Padrões de Software 6
Memento 6
Singleton 6
Observer 7
Simple Factory 7
Data Access Object 8
Refactoring 9
Após Refactoring 9
Duplicated Code 9
Long Method 10
Inline Method - Exemplo 11

TADs implementadas

Queue

Aplicada no algoritmo BFS

Stack

- Aplicada no algoritmo DFS
- Usado no CareTaker para guardar os vários estados do Digrafo

DiGraph

Usado como TAD base do projeto

Diagrama de classes

Ver imagem “uml.png” para poder visualizar melhor.

Classes

projecto.model

Interface Summary	
Interface	**Description**
Memento	Interface que guarda o estado atual do Digrafo e do caminho
Observer	Interface Observer
Originator	Interface Originator
WebCrawlerDAO	Interface para definir as operações que se pretende efetuar sobre o objecto

Class Summary	
Class	**Description**
Link	Classe responsável por armazenar a informação de cada link
MementoWebCrawler	Classe que representa o Memento da classe WebCrawler
Page	Classe responsável por armazenar a informação de cada página
WebCrawlerAction	Classe responsável pela criação dos tipos de ficheiros
WebCrawlerCareTaker	Classe que representa o CareTaker
WebCrawlerJSon	Classe que realiza as ações do DAO em formato JSON
WebCrawlerSerialization	Classe que realiza as ações do DAO em tipo serialização
WebLogger	Classe Singleton que representa o Log de quando é gerado um grafo

Enum Summary	
Enum	**Description**
Page.PageRole	Enum para distinguir a main page das restantes

Exception Summary	
Exception	**Description**
WebActionException	Classe de exceções relativamente á classe WebAction
WebCrawlerException	Classe que lança exceções relacionadas com a classe WebCrawler

projecto

Class Summary	
Class	**Description**
Main	Classe javafx que representa o menu inicial da aplicação
MainPage	Main classe que pergunta ao utilizador qual irá ser a main page para executar o programa
ModalWindow	Classe javafx que representa um modal ao qual o utilizador pode inserir o número máximo de pages ou links, consoante o algoritmo selecionado
WebCrawler	Classe que, é feita a gestão geral do Digrafo e suas estatística.

com.brunomnsilva.smartgraph.containers

Class Summary	
Class	**Description**
ContentResizerPane	
ContentZoomPane	This class provides zooming and panning for a JavaFX node.
SmartGraphDemoContainer	Classe javafx que mostra um Digrafo gerado automaticamente
SmartGraphDemoContainerIterativo	Classe javafx que mostra um Digrafo gerado iterativamente
SmartGraphDemoContainerPath	Classe javafx que mostra um caminho do Digrafo

Restantes classes, podem ser consultadas em formato *javadoc*, abrindo o ficheiro index.html na pasta *javadoc* fornecida dentro do projeto.

Padrões de Software

Memento

- **Finalidade**
  - Guardar o estado de um objecto, sem violar o encapsulamento do mesmo, para que seja possível retornar a esse estado em qualquer altura.

- **Implementação no Projeto**
  - A utilização do Memento permitiu a funcionalidade de "Undo", isto é, desfazer algo que foi feito. Foi implementado de forma a permitir ao utilizador desfazer a seleção de uma ou mais paginas.

- **Classes do Padrão**
  - Memento
    - MementoWebCrawler
  - Originator
    - WebCrawler
  - CareTaker
    - WebCrawlerCareTaker

Singleton

- **Finalidade**

  Garantir que uma classe só tenha uma única instância e criar um ponto de acesso global a essa instância.

- **Implementação no Projeto**
  - Utilizado por uma classe de maneira a fornecer, uma forma consistente sem sobrecarregar a aplicação.

- **Classes do Padrão**
  - Singleton
    - WebLogger

Observer

- **Finalidade**
  - O padrão Observer permite que objectos interessados sejam avisados da mudança de estado ou de outros eventos ocorridos num outro objecto.

- **Implementação no Projeto**
  - A utilização do Observer permitiu atualizar os dados do Digrafo na parte gráfica, como consequência, as estatísticas estão sempre atualizadas (Modo iterativo)

- **Classes do Padrão**
  - Subject
    - WebCrawler
  - Observer
    - Observer
  - Concrete Observer
    - SmartGraphDemoContainerIterativo

Simple Factory

- **Finalidade**
  - Quando se tem várias classes que implementam uma interface (ou uma hierarquia de clases), e se pretende delegar a responsabilidade para criar um objeto num outro objecto, encapsulando as diversas implementações da interface.

- **Implementação no Projeto**
  - A utilização do Simple Factory foi usado na parte de que o utlizador escolhe qual é a forma que quer exportar/importar o Digrafo.

- **Classes do Padrão**
  - Product
    - WebCrawlerDAO
  - Factory
    - WebCrawlerAction
  - Concrete Product
    - WebCrawlerJSon
    - WebCrawlerSerialization

Data Access Object

- **Finalidade**
  - Forma de aceder aos dados.

- **Implementação no Projeto**
  - A utilização do DAO do projeto baseou-se na parte da Persistência dos dados, onde o utilizador pode importar/exportar os dados do Digrafo.

- **Classes do Padrão**
  - DAO Interface
    - WebCrawlerDAO
  - DAO Concrete Class
    - WebCrawlerSerialization
    - WebCrawlerJSon
  - Model Object or Value Object
    - WebCrawler

Refactoring

Bad-smells	Ocorrências	Técnicas usadas
Duplicated Code	11	Extract Method, Inline Method
Long Method	4	Extract Method

Após Refactoring

Duplicated Code

- **Antes**

```
SeriaButton.setOnAction(e -> {

    if (checkFile("serialization") == null) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("Import");
        alert.setHeaderText(null);
        alert.setContentText("Não existe Digrafo guardado!");
        alert.showAndWait();
    } else {
        generateDiGraph(checkFile("serialization"));

        ((Stage) bfsButton.getScene().getWindow()).close();
    }

});

JsonButton.setOnAction(e -> {

    if (checkFile("json") == null) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("Import");
        alert.setHeaderText(null);
        alert.setContentText("Não existe Digrafo guardado!");
        alert.showAndWait();
    } else {
        generateDiGraph(checkFile("json"));

        ((Stage) bfsButton.getScene().getWindow()).close();
    }
});
```

- **Depois**

```
SeriaButton.setOnAction(e -> {

    if (checkFile("serialization") == null) {
        showAlert();
    } else {
        generateDiGraph(checkFile("serialization"));

        ((Stage) bfsButton.getScene().getWindow()).close();
    }

});

JsonButton.setOnAction(e -> {

    if (checkFile("json") == null) {
        showAlert();
    } else {
        generateDiGraph(checkFile("json"));

        ((Stage) bfsButton.getScene().getWindow()).close();
    }
});
```

Fix

```
public static void showAlert() {
    Alert alert = new Alert(Alert.AlertType.ERROR);
    alert.setTitle("Import");
    alert.setHeaderText(null);
    alert.setContentText("Não existe Digrafo guardado!");
    alert.showAndWait();
}
```

Long Method

- **Antes**

```
for (Element elem : ele) {

    String url = elem.attr("abs:href");
    String label = elem.text();

    if (!label.contains("404")) {
        if (urlNotEmpty(url)) {
            Document titlePage = Jsoup.connect(url).get();
            if (titlePage != null) {
                String titulo = titlePage.title();

                if (!label.isEmpty()) {
                    Page pp = getPagebyTitle(titulo);
                    if (!visited.contains(pp)) {
                        if (pp == null || !havePage(pp)) {
                            Page newPage = new Page(titulo, Page.PageRole.SUB);
                            addPage(newPage);
                            Vertex<Page> VPage = getPage(newPage);

                            Link link1 = new Link(url, label);
                            addLink(link1, pagina, VPage.element());

                            WebLogger.getInstance().saveToLog(pagina, newPage, link1, this);

                            queue.offer(newPage);
                            pages.add(getPage(newPage));
                            visited.add(newPage);
                        }
                    } else if (!pagina.getTitle().equals(pp.getTitle()) && !LinkExistBetweenPages(pagina, pp, url)) {
                        Link link1 = new Link(url, label);
                        addLink(link1, pagina, getPagebyTitle(titulo));

                        WebLogger.getInstance().saveToLog(pagina, pp, link1, this);
                    }
                } else {
                    Page p1 = new Page("404 - Not Found", Page.PageRole.SUB);
                    addPage(p1);
                    Link link1 = new Link(url, "Not found");
                    addLink(link1, pagina, p1);

                    WebLogger.getInstance().saveToLog(pagina, p1, link1, this);
                }
            }
        }
    } else {
        Page p1 = new Page("404 - Not Found", Page.PageRole.SUB);
        addPage(p1);

        Link link1 = new Link(url, "Not found");
        addLink(link1, pagina, p1);

        WebLogger.getInstance().saveToLog(pagina, p1, link1, this);
    }
}
```

- **Depois**

```
for (Element elem : ele) {

    String url = elem.attr("abs:href");
    String label = elem.text();

    if (!label.contains("404")) {
        if (urlNotEmpty(url)) {
            Document titlePage = Jsoup.connect(url).get();
            if (titlePage != null) {
                String titulo = titlePage.title();

                if (!label.isEmpty()) {
                    Page pp = getPagebyTitle(titulo);
                    if (!visited.contains(pp)) {
                        if (pp == null || !havePage(pp)) {
                            generateBFS(visited, pages, queue, pagina, titulo, url, label);
                        }
                    } else if (!pagina.getTitle().equals(pp.getTitle()) && !LinkExistBetweenPages(pagina, pp, url)) {
                        Link link1 = new Link(url, label);
                        addLink(link1, pagina, getPagebyTitle(titulo));

                        WebLogger.getInstance().saveToLog(pagina, pp, link1, this);
                    }
                } else {
                    createPageNotFound(pagina, url);
                }
            }
        }
    } else {
        createPageNotFound(pagina, url);
    }
}
```

Inline Method - Exemplo

- **Antes**

```java
/**
 * *
 * Retorna o número total de relações que uma página contem
 *
 * @param v Recebe um vertice de uma página
 * @return Retorna o número de relações de uma página
 */
public int getNumberofRelationships(Vertex<Page> v) {
    return getAdjacent(v).size();
}

/**
 * *
 * Retorna a página com mais ligações
 *
 * @return Pagina que tem o maior número de relações
 */
public Page getMostLinkedPage() {

    Page popular = null;
    int maxRelationships = 0;

    for (Vertex<Page> v : web.vertices()) {
        int nRela = getNumberofRelationships(v);
        if (nRela > maxRelationships) {
            maxRelationships = nRela;
            popular = v.element();
        }
    }
    return popular;
}
```

- **Depois**

```java
/**
 * *
 * Retorna a página com mais ligações
 *
 * @return Pagina que tem o maior número de relações
 */
public Page getMostLinkedPage() {

    Page popular = null;
    int maxRelationships = 0;

    for (Vertex<Page> v : web.vertices()) {
        int nRela = getAdjacent(v).size();
        if (nRela > maxRelationships) {
            maxRelationships = nRela;
            popular = v.element();
        }
    }
    return popular;
}
```