|  |
| --- |
| Programação Avançada  2018/2019  Licenciatura em Engª. Informática |
| Aplicação para Emissão de Bilhete-Percurso num Parque Biológico |



Docente: Bruno Silva

Realizado por:

140221053 - Jorge Silva

150221006 - João da Silva Gomes

Índice

[Introdução 3](#_Toc536480580)

[TADs implementadas 4](#_Toc536480581)

[Diagrama de classes 5](#_Toc536480582)

[Javadoc 9](#_Toc536480583)

[Padrões de software 10](#_Toc536480584)

[Refactoring 12](#_Toc536480585)

[Libraries/JarFiles utilizados 28](#_Toc536480593)

[Tecnologias utilizadas 29](#_Toc536480598)

# Introdução

Pretende-se desenvolver uma aplicação que permita gerar percursos a pé e de bicicleta dentro de um parque biológico. A aplicação disponibiliza informação sobre o preço total a pagar pelo percurso selecionado e permite ainda a emissão de bilhetes e respetiva fatura.

O parque biológico é constituído por vários pontos de interesses, pontos esses que estão conectados ou por caminhos, ou por pontes; os caminhos podem ser percorridos em qualquer sentido, mas as **pontes só podem ser percorridas num sentido**. Existem **ainda conexões que não permitem a circulação de bicicletas**. Cada conexão tem um custo, e uma distância associada.

Cada percurso **inicia e termina no ponto de entrada do parque**, e poderá passar por vários pontos de interesse selecionados pelo utilizador. O utilizador deverá poder calcular o seu percurso de forma a minimizar a distância, ou o custo do mesmo.

# TADs implementadas

No nosso projeto tivemos de recorrer à utilização do algoritmo Dijkstra, este calcula o custo mínimo do vértice raiz para os demais. Assim sendo, usámos este algoritmo sobre um dígrafo orientado com a raiz no nosso ponto de entrada, os restantes pontos do nosso mapa sendo os vértices do grafo e os caminhos/pontes as diversas arestas orientadas.

# Diagrama de classes

1. BILHETE

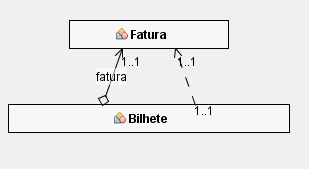


Fig.1

1. DAO

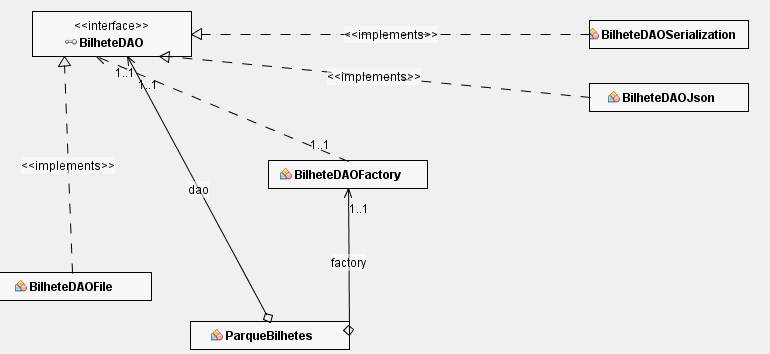


Fig.2

1. MVC CONTROLLER



Fig.3

1. MVC MODELS

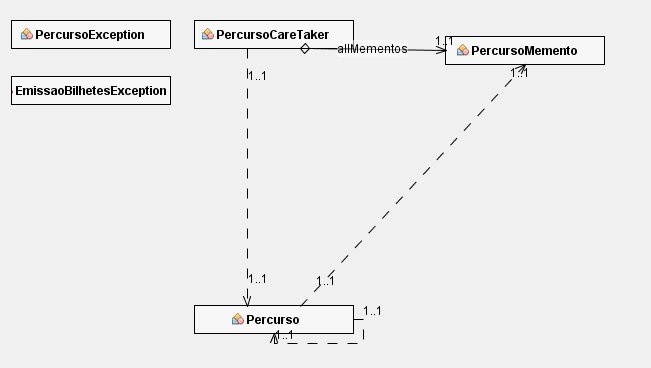


Fig.4

1. MVC VIEWS

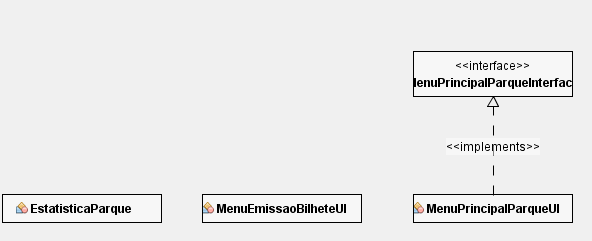


Fig.5

1. PDFGenerator



Fig.6

1. SINGLETON

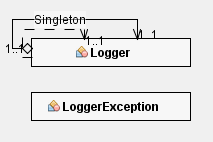


Fig.7

1. MAIN



Fig.8

1. DIGRAPH GRAPH

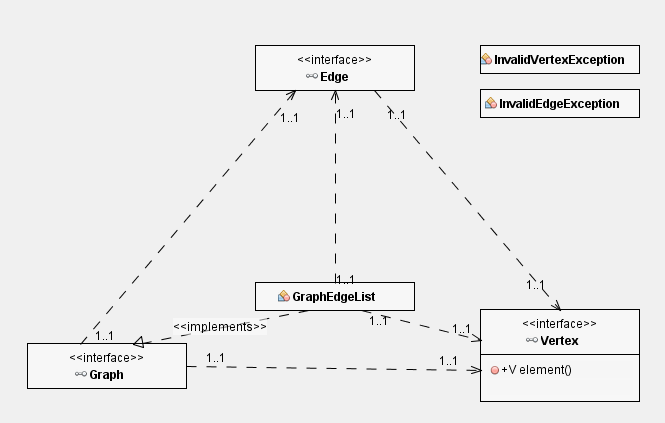


Fig.9

1. DIGRAPH MODEL

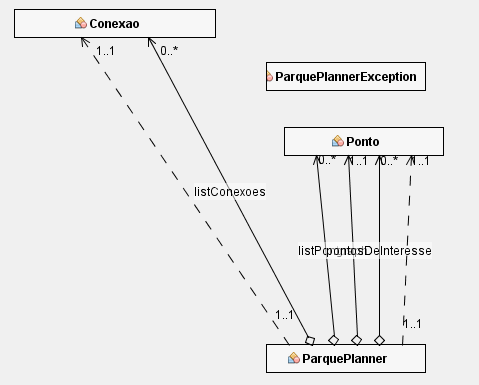


Fig.10

# Javadoc

**Bilhete** - Classe que representa um bilhete com a informação do percurso associado.  
**Fatura** - Classe que representa uma fatura associada a um bilhete.  
**BilheteDAOFactory** - Classe que cria instâncias de bilhetes.  
**BilheteDAOFile** - Classe File onde se cria as tabelas manualmente para inserir o bilhete.  
**BilheteDAOJson**- Classe que guarda o bilhete como objecto através da sua coleção de pares de valores.  
**BilheteDAOSerialization** - Serialização de uma classe, é ativada pela classe que implementa a interface do bilhete.  
**PercursoController** - Classe controller do percurso.  
**Percurso** – Classe model para o percurso.  
**PercursoCareTaker** - Classe Care Taker para guardar e recuperar os estados do percurso.  
**PercursoMemento** - Classe Memento do MVC.  
**EstatisticaParque** - Class View das estatísticas do parque.  
**MenuEmissaoBilheteUI** - Classe view do menu de emissão de bilhete.  
**MenuPrincipalParqueUI** - Classe view do menu principal do parque.  
**GenerateTicket** - Classe que gera os PDF's de fatura e bilhete.  
**Conexao** - Classe modelo do algoritmo Dijkstra que gere o percurso atual com todos os pontos de interesse.  
**ParquePlanner** - Classe modelo que gere o parque.  
**Ponto** - Classe representativa de um ponto do grafo.

# Padrões de software

|  |  |  |
| --- | --- | --- |
| Nome | Definição | Utilização |
| Data Acess Object  (DAO) | DAO é um padrão de projetos onde um objeto:   * provê uma interface que abstrai o acesso a dados; * lê e grava a partir da origem de dados (banco de dados, arquivo, memória, etc.); * encapsula o acesso aos dados, de forma que as demais classes não precisam saber sobre isso. | Padrao que guarda o Bilhete como objecto, e todos os seus atributos.  Tipos Implementados:   1. BilheteDAOFile 2. BilheteDAOJson 3. BilheteDAOSerialization |
| Model View Controler  (MVC) | P[adrão de arquitetura](https://pt.wikipedia.org/wiki/Padr%C3%A3o_de_arquitetura) de software que separa a representação da informação da interação do usuário com ele. Este [padrão separa](https://pt.wikipedia.org/wiki/Padr%C3%A3o_de_projeto) componentes maiores possibilitando a [reutilização de código](https://pt.wikipedia.org/wiki/Reutiliza%C3%A7%C3%A3o_de_c%C3%B3digo) e desenvolvimento paralelo de maneira eficiente. | Componentes Principais  Models:   * Percurso   Views:   * MenuPrincipalParqueUI * MenuEmissaoBilheteUI   Controller:   * PercursoController   MVC é uma aqutectura entre camadas em que no nosso caso, o percurso é o modelo, que é controlado pelo percursoController que por sua vez tem as views representadas. |
| Observer | [Padrão de projeto de software](https://pt.wikipedia.org/wiki/Padr%C3%A3o_de_projeto_de_software) que define uma dependência um-para-muitos entre objetos de modo a que quando um [objeto](https://pt.wikipedia.org/wiki/Objeto) muda o estado, todos os seus dependentes são notificados e atualizados automaticamente. | Observable: ParquePlanner  Observer: MenuPrincipalParqueInterface    O modelo parquePlanner é o modelo que está a ser observado e manipulado pela view que é o observador |
| Singleton | P[adrão de projeto de software](https://pt.wikipedia.org/wiki/Padr%C3%B5es_de_projeto_de_software) que garante a existência de apenas uma instância de uma [classe](https://pt.wikipedia.org/wiki/Classe_(programa%C3%A7%C3%A3o)), mantendo um ponto global de acesso ao seu [objeto](https://pt.wikipedia.org/wiki/Objeto_(programa%C3%A7%C3%A3o)). | Visto que na aplicação precisamos de uma infraestrutura de log de dados, implementamos assim uma classe do padrão singleton. Desta forma existe apenas um objeto responsável pelo logger. |
| Memento | Padrão de projeto de software que permite armazenar o estado interno de um objeto em um determinando momento, para que seja possível retorná-lo a este estado, sem que isso cause problemas com o encapsulamento. | Utilizamos para guardar o estado da lista de pontos de interesse escolhida pelo utilizador |

# Refactoring

|  |  |  |
| --- | --- | --- |
| Tipos de Bad-smells | Número de situações | Técnicas de Refactoring |
| Duplicated Code | 2 | Extract Method, Extract Class, Pull Up Method, Form Template Method |
| Speculative Generality | 1 | Collapse Hierarchy, Inline Class, Remove Parameter, Rename Method |
| Long Method | 3 | Extract Method, Replace Temp with Query |
| Switch Statments | 1 | Replace Conditional with Polymorphism, Replace Type Code with Subclasses, Replace Type Code with State/Strategy, Replace Parameter with Explicit Methods, Introduce Null Objects |
| Dead Code | 5 | Delete unused code and unneeded files |
| Temporary Field | 1 | Extract Class, Introduce Null Object |

|  |  |
| --- | --- |
| Bad smell |  |
| Explicação |  |
| Técnica de refactoring |  |
| Antes refactoring |  |
| Após refactoring |  |

|  |  |
| --- | --- |
| Bad smell | Speculative Generality |
| Explicação | Quando existe uma classe, um método ou um parâmetro não utilizado.  Neste caso estávamos a fazer o try catch mal não passando a messagem do erro. |
| Técnica de refactoring | Introduce Assertion- Pedaço de código assume alguma coisa sobre o estado do programa. Fazer com que a assunção seja explicita consoante a asserção. |
| Antes refactoring |  |
| Após refactoring |  |

|  |  |
| --- | --- |
| Bad smell | Long method ; Switch Statments |
| Explicação | Long method-Um método com muitas linhas de código;  Switch Statments- Quando se possui um operador switch ou uma sequência de instruções if. |
| Técnica de refactoring | Replace Error Code with Exception- Método que retorna mensagem de erro com em vez disso retorna a exceção e o tratamento dela com a View.showError(). |
| Antes refactoring |  |
| Após refactoring |  |

|  |  |
| --- | --- |
| Bad smell |  |
| Técnica de refactoring | Replace Error Code with Exception |
| Antes refactoring |  |
| Após refactoring |  |

|  |  |
| --- | --- |
| Bad smell |  |
| Técnica de refactoring | Replace Error Code with Exception |
| Antes refactoring |  |
| Após refactoring |  |

|  |  |
| --- | --- |
| Bad smell | Dead Code - Linha de código que nunca é utilizada |
| Explicação | If dentro de try catch , impossibilidade de capturar o erro e trata-lo, Assim retiramos o if e tratamos o erro com a view. |
| Técnica de refactoring | Move Method- movemos a captura de erro no getBtEmitirBilhete |
| Antes refactoring |  |
| Após refactoring |  |

|  |  |
| --- | --- |
| Bad smell |  |
| Técnica de refactoring | Move Method |
| Explicação | Removemos o ciclo *for* dentro do contructor e criamos o metodo que é responsavél por adicionar pontos à comboBox. |
| Antes refactoring |  |
| Após refactoring |  |

|  |  |
| --- | --- |
| Bad smell | Dead Code |
| Explicação | PercursoManager *class* que se tornou obsoleta na nossa implementação, pois não iremos alterar o percurso final depois de criado. |
| Técnica de refactoring | *Class* que não é mais utilizada pois tornou-se obsoleta. |
| Antes refactoring |  |
| Após refactoring | Eliminada a classe inutilizada. |

|  |  |
| --- | --- |
| Bad smell | Dupicated code |
| Explicação | Definir as condiçoes no metodo getNifInput() para capturar e tratar os erros na acção do generateBilhete() |
| Técnica de refactoring | Extract Mehod-Mover o codigo para um novo metodo separado e substituir o código por uma chamada para o novo metodo |
| Antes refactoring |  |
| Após refactoring |  |

|  |  |
| --- | --- |
| Bad smell | Long Method |
| Explicação | Ao observarmos o metodo start do main, conseguimos ver bastante desorganização e um metodo bastante longo, o que nos levou a fragmentar o codigo |
| Técnica de refactoring | Extract Method- Mover o codigo para um novo metodo separado e substituir o código por uma chamada para o novo metodo |
| Antes refactoring |  |
| Após refactoring |  |

|  |  |
| --- | --- |
| Bad smell | Long Method |
| Explicação | O metodo estava confuso e demasiado longo logo fragmentamos o codigo e fizemos uma chamada aos novos metodos(*setPathsColor(planner, graphView); setArrows(graphView);)* no *start()* |
| Técnica de refactoring | Extract Method- Mover o codigo para um novo metodo separado e substituir o código por uma chamada para o novo metodo |
| Antes refactoring |  |
| Após refactoring |  |

|  |  |
| --- | --- |
| Bad smell | Dead Code |
| Explicação | Excluir o código não utilizado |
| Técnica de refactoring | Metodo que não é mais utilizado pois tornou-se obsoleto. |
| Antes refactoring |  |
| Após refactoring | Eliminado o método inutilizado. |

|  |  |
| --- | --- |
| Bad smell | Dead Code |
| Explicação | *Class* que se tornou obsoleta ao nosso padrão de M.V.C. |
| Técnica de refactoring | Class que não é mais utilizada pois tornou-se obsoleta |
| Antes refactoring |  |
| Após refactoring | Eliminada a classe inutilizada. |

|  |  |
| --- | --- |
| Bad smell | Temporary Field |
| Explicação | Campos temporários são criados para uso em um algoritmo que requer uma grande quantidade de entradas. Portanto, em vez de criar um grande número de parâmetros no método, decidimos criar campos para esses dados na classe. Esses campos são usados ​​apenas no algoritmo e não são usados ​​o resto do tempo. |
| Técnica de refactoring | Criar atributos de class |
| Antes refactoring |  |
| Após refactoring |  |

|  |  |
| --- | --- |
| Bad smell | Dead Code-Unused Parameter |
| Explicação | Reparámos que existiam vários imports desnecessários e inutilizados |
| Técnica de refactoring | Remove Parameter |
| Antes refactoring |  |
| Após Refactoring |  |

|  |  |
| --- | --- |
| Bad smell | Duplicated Code |
| Explicação |  |
| Técnica de refactoring |  |
| Antes refactoring | https://scontent.flis7-1.fna.fbcdn.net/v/t1.15752-9/50636742_293163234679010_2211868222834081792_n.png?_nc_cat=101&_nc_ht=scontent.flis7-1.fna&oh=16bf2056fefe1c480a94145ccc7dc417&oe=5CC4F6E0 |
| Após refactoring |  |

# Libraries/JarFiles utilizados

# Estão na pasta de Jar\_Files:

# gson-2.3.1 - Utilizado para o DAO BilheteDAOJson

# itextpdf-5.5.4 - Utilizado gerar o pdf e o QR code

# sqlite-jdbc-[3.21](https://www.facebook.com/).0 - Utilizado para o DAO

# Tecnologias utilizadas

[http://graphonline.ru/en/](http://graphonline.ru/en/?fbclid=IwAR2E9nk_6ADo2Tk2YvtT66nsl--4JA1FU_XSkt4cognZAA4Xqf_DNVMXBBU) - Utilizado para auxilio de calculo e teste de percursos calculados.