

**Relatório**



Trabalho realizado por:

João Pedro Martins nº160221033  
David Tavares nº160221046

# Índice

* [Introdução](#_Introdução)
* [Especificação das TADs implementadas](#_TADs_implementadas)
* Diagrama de classes
* Documentação das classes
* Descrição/uso/justificação dos padrões de software utilizados
* Refactoring
* Conclusão

# Introdução

O objetivo deste projeto consiste no desenvolvimento de uma aplicação de desktop para geração de percursos num parque biológico e respetiva emissão de bilhete.

Utilizamos o IDE NetBeans para a realização de tal projeto e foi desenvolvido na linguagem Java.

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.

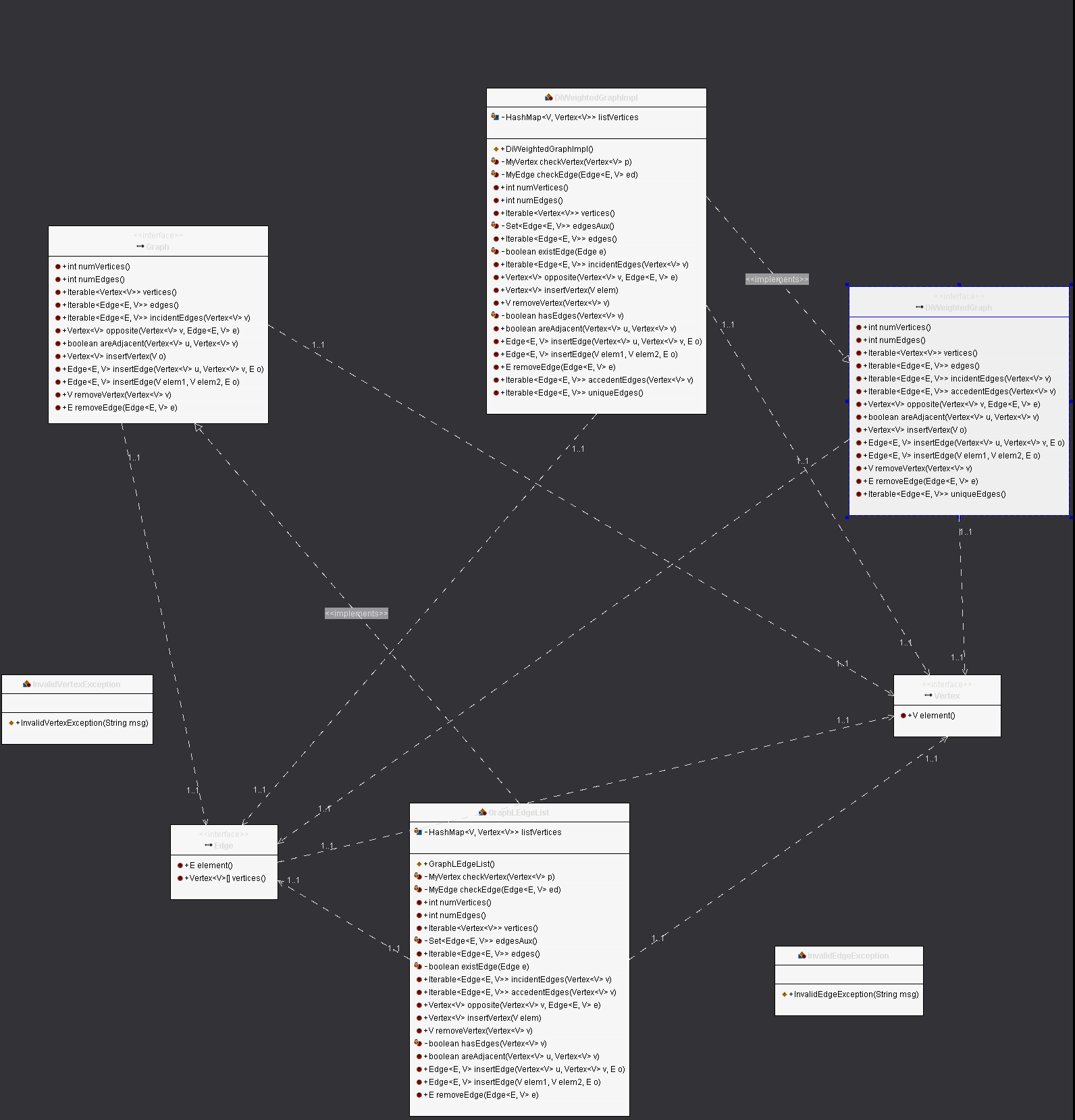
Cada percurso inicia e termina no ponto de entrada do parque, e poderá passar por vários pontos de interesse selecionados pelo utilizador.

O utilizador deverá poder calcular o seu percurso de forma a minimizar a distância, ou o custo do mesmo.

# TADs implementadas

* Digrafos
* Tree

# Diagrama de classes



# Documentação das classes

**Dot,** esta classe representa um ponto no mapa.

**Route**, esta classe representa uma aresta no mapa.

**RouteType,** esta classe tem como objetivo identificar se a aresta em questão é uma ponte ou um caminho.

**Arrow,** esta classe representa uma seta na cubicCurve.

**CircularPlacementStrategy,** estratégia que dispõe os pontos(vértices) numa forma circular.

**CircularSortedPlacementStrategy**, estratégia que dispõe os pontos(vértices) numa forma circular mas por ordem alfabética.

**GraphEdge,** uma aresta arrastável na forma de cubicCurve

**GraphPanel,** uma classe JavaFX pane que é capaz de desenhar o grafo, pintar os pontos/arestas.

**GraphVertex,** um vértice arrastável representado na forma de um circulo.

**RandomPlacementStrategy,** estratégia que dispõe os pontos(vértices) de forma aleatória.

**UtilitiesPoint2D,** classe com os métodos uteis para as instancia de pontos2D.

**VertexPlacementStrategy**, Interface que descreve a estratégia de disposição dos pontos no GraphPanel.

**Billing,** esta classe representa a interface de uma fatura e para o bilhete, tem como objetivo imprimir para um ficheiro pdf as informações sobre o percurso selecionado.

**BillingNIF,** esta classe representa uma fatura com contribuinte.

**RegularBilling,** esta classe irá emitir um fatura sem contribuinte e um bilhete.

**CourseInfo,** esta classe guarda todas as informações necessárias para calcular um percurso.

**CourseManager,** responsável por ler o mapa, por calcular os percursos e por criar o grafo.

**Memento,** classe que guardará as informações de um caminho para realizar o undo esta classe aplica o padrão Memento para guardar e recuperar "states" dos percursos.

**Caretaker,** interface realizada com base no padrão memento, com o objetivo de aplicar o undo, restaura e salva "states".

**Controler,** classe responsável pela lógica das views e do model.

**MainPane**, classe que representa a View principal do programa.

**MakePath,** Classe que representa a view mostrada quando clicamos no botão "fazer percurso".

**MapSelection,** esta classe é responsável por inicializar o primeiro menu, onde será escolhido o mapa, a persistência e o logger .

**Stats,** esta classe é responsável por mostrar o menu das estatísticas com os devidas tabelas.

**TicketMenu**, Esta classe é representa todas a views mostradas apos ser clicado no botão emitir bilhete da view MainPane.

**TopBar,** esta classe representa a MenuBar do menu principal.

**DAOCourse**, Interface que aplica o padrão DAO para escolher a persistência.

**DAOSerializationCourse**, Classe para persistir os dados em serialização.

**Logg,** Esta classe é responsável por registar as informações da aplicação num ficheiro Logger.txt

**PercentageStats**, Esta classe é responsável por fazer os cálculos das percentagens de cada tipo de percurso (a pé, ou de bicicleta)

# Descrição/uso/justificação dos padrões de software utilizados

Padrões utilizados:

* Memento
* DAO pattern
* Singleton
* Template

Memento

Descrição:

Procuramos com este padrão guardar um estado da classe CourseInfo mas afetar o encapsulamento da mesma, desta forma podemos restaurar a qualquer momento esse mesmo estado.

Uso:

Classes influenciadas pelo o padrão: Memento, Caretaker, CourseInfo(originador).

Justificação:

Utilizamos este padrão com a finalidade de criar um botão undo no programa onde o utilizador possa a qualquer altura restaurar os seus percursos anteriores.

Singleton

Descrição:

Procuramos com este padrão garantir que a classe Logg tenha apenas uma única instância e criar um ponto de acesso global a essa instancia.

Uso:

Classe influenciadas pelo padrão: Logg

Justificação:

Através deste padrão garantimos que todas as informações sobre consultas ao percurso, estatísticas e emissão de bilhete são guardadas no mesmo ficheiro logger.

DAO

era podermos utilizar serialization e SQL como persistência.

Template

Descrição/uso:

Temos uma superclasse Billing que tem 2 subclasses, em função do tipo de Bilhete: BillingNIF, RegularBilling.

Justificação:

Visto termos algoritmos idênticos para a emissão de bilhete e faturas, optámos por aplicar o padrão template para as subclasses BillingNIF e RegularBilling.

DAOpattern

Descrição/justificação:

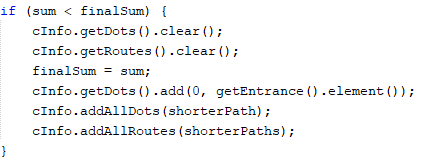
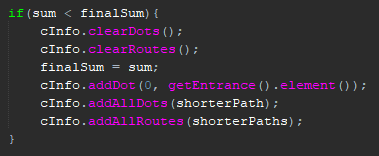
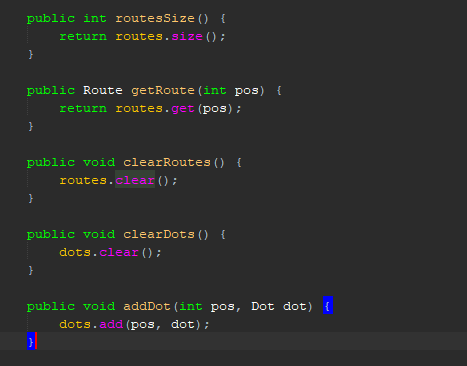
A forma de aceder aos dados depende da fonte dados. Logo a componente de acesso aos dados depende do tipo de fonte de dados utilizada. Para isso utilizamos o padrão DAO para abstrair e encapsular todos os acessos à fonte de dados independentemente de ser uma base de dados ou um ficheiro.

Uso:

Criamos a interface DAOcourse que define as operações standart, ou seja, é o participante DAO interface, em seguida criamos os participantes DAO concrete Classes, DAOserilizationCourse e DAOsqlLite(não implementada), que implementam a interface e são as classes responsáveis por ir buscar os dados à entidade responsável pela sua persistência.

# Refactoring

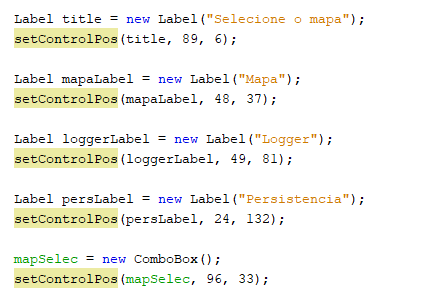
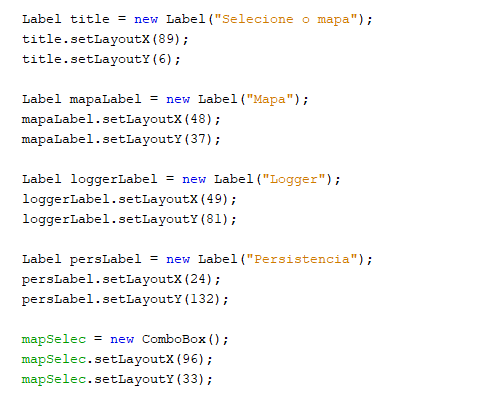
|  |  |  |
| --- | --- | --- |
| Exemplo: | Bad-smell | Técnica |
| 1 | Inappropriate Intimacy | Hide delegate |
| 2 | Data Class | Encapsulate collection |
| 3 | Duplicated code | Extract method |
| 4 | Message Chain | Extract method |



depois

antes

1

 Restruturação da classe courseInfo,

2

Antes: era uma simples data class

Depois: adição de métodos que não apenas gets e sets.

3

Antes: depois: