Relatório PA – Fase Final

ANDRÉ GONÇALVES 170221015
DUARTE GONÇALVES 170221028

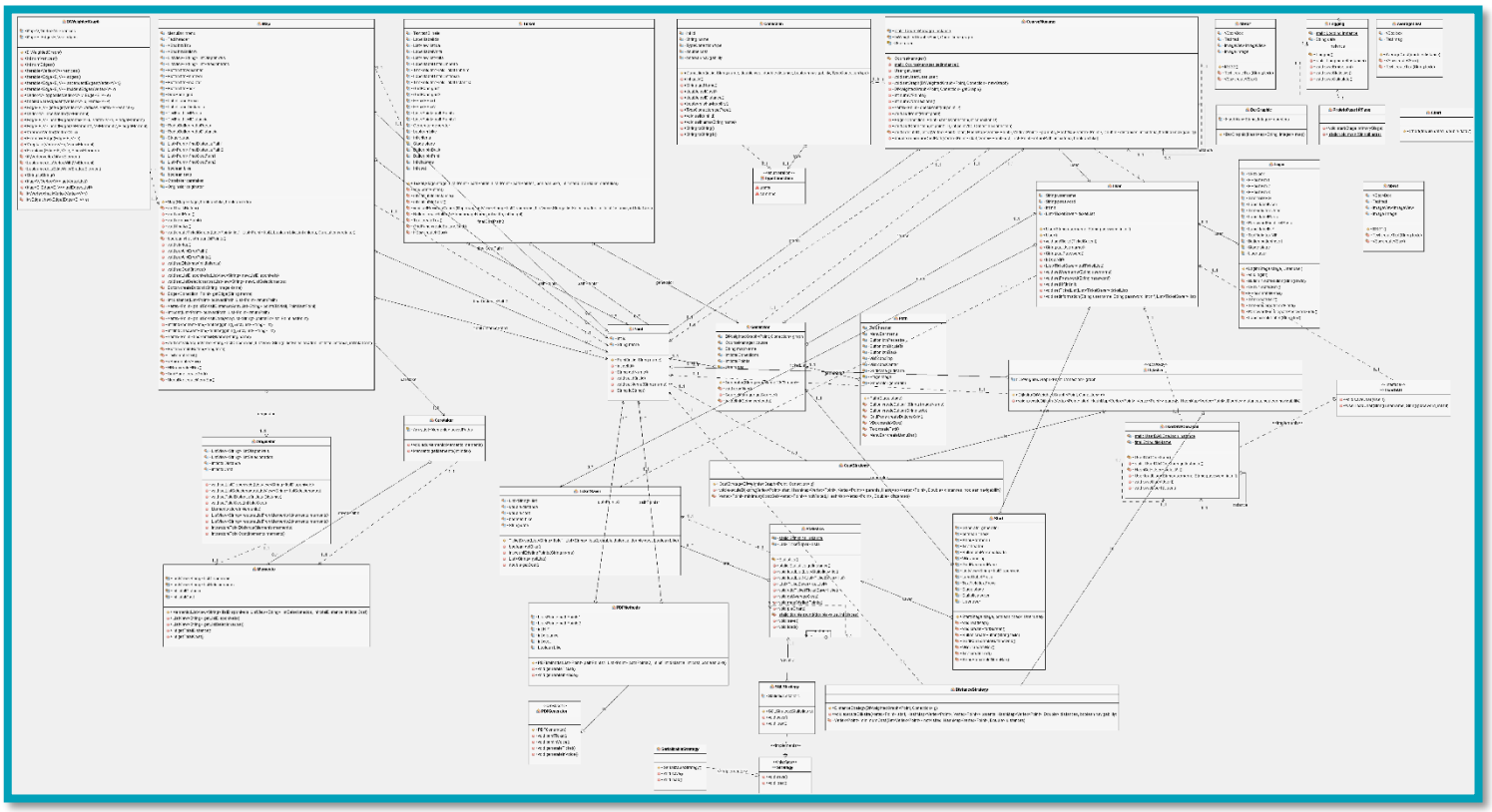
Índice

TAD's implementados

DIWEIGHTEDGRAPH

Grafo que contém arestas com pesos e distâncias. Ideal na representação de mapas com diversas modalidades de deslocação.

Diagrama de classes



Documentação de Classes

Class Summary

Class	Description
DiWeightedGraph<V,E>	ADT Graph implementation that stores a collection of edges (and vertices) and where each edge contains the references for the vertices it connects.

Class Summary

Class	Description
AverageCost	Class that will create a screen to show the average cost of ticket cost that have already been created
BarGraphic	Class that will create a screen with a bar graphic with the number of points visited
Chart	Class that will create a screen with a Pie Chart with the percentage of bike ridden tickets and not bike ridden tickets.
Login	Class that creates the interface for the user to login, by inserting its username, password and NIF code (if wanted)
Map	
MSG1	Class that creates a Pop-Up Screen to inform the user that he can't choose the path he wants because its unaccesible by Bike
MSG2	Class that creates a Pop-Up Screen to inform the user that he can't finalize the ticket whitout inserting at least 1 point to visit without counting the Entrance.
Path	Class that will create a screen that will allow the user to choose if he wants to use a bike or not.
Start	Screen that will let the user choose if he wants to choose the path by himself or if he wants a premade one.
Ticket	Class that will show the outward and return path with the points that user want to visit and its total distance or cost.

Class Summary

Class	Description
Caretaker	Class that allows to set and retrive the Memento values.
Conection	Class that represents the Edges of the DiWeightedGraph.
CostStrategy	Class that creates one of the strategies for the djikstra method using the cost as a criteria.
CourseManager	This class deals with mosth of the Graph values and executes the djikstra using a Strategy Pattern by using the criteria.
Dijkstra	Class that represents the Djikstra method to use different types of this method.
DistanceStrategy	Class the implements the Djikstra method using the distance as it's criteria.
Generator	Class that will read the .dat file and will insert the point's and conection's to the graph for later use.
Logging	Class the will proovide the register the activity of the user durint his use of the application.
Memento	Class to save and retrive the previous values of certain elements.
Originator	Changes the values of the variables inside of the memento.
PDFGenerator	Abstract class that will call the different methods to print the receipt and the invoice.
PDFMethods	Class that contains the methods to create the PDF for the receipt and the invoice.
Point	Class that represents the Vertices of the graph.
SerializableStrategy	Class that will save the values of the application in .bin file.
SQLStrategy	Class that will save the values of the application in a SQL Database.
Statistics	This class will handle the statistics side of the applciation.
TicketSaver	This class is the representaion of a ticket.
User	This class represents the user using the application.
UserDAOOneJson	This class handles the user's .json file.

Descrição/uso/justificação dos padrões de software utilizados

SINGLETON

Utilizado no Gestor de Percursos, Logging, Statistics e UserDAO. No Gestor de Percursos utilizámos o padrão Singleton pois apenas existe uma instância do mesmo (vendedor de bilhetes). O Logging requer a utilização do padrão Singleton pois existe apenas um "gestor/escritor" de atividade. As Statistics requerem a utilização do padrão Singleton porque são estatísticas gerais aplicadas ao vendedor de bilhetes, ou seja, apenas 1 instância. O UserDAO também consiste apenas numa instância que guarda os utilizadores (compradores de bilhetes).

STRATEGY

O padrão Strategy foi utilizado para guardar as Statistics e também para o a escolha do caminho mais curto/barato com o algoritmo Dijkstra. No caso das Statistics, é utilizado de modo a diferenciar a forma de guardar e ler os dados através de java persistence(serializable) ou na base de dados SQLite. Para a escolha do caminho por parte do utilizador, a Strategy é utilizada quando este decide qual o tipo de caminho que deseja: mais curto ou mais barato.

TEMPLATE

O padrão Template foi utilizado para a emissão de bilhete e fatura, de forma a evitar a duplicação de código na criação dos 2 documentos.

DAOPATTERN

O DAOPattern foi o padrão utilizado na gestão de utilizadores (compradores) para a simulação de venda de bilhetes. Neste caso, é utilizado para guardar utilizadores e posteriormente "chamá-los" quando necessário (login feito na máquina de venda).

MEMENTO

O memento foi utilizado para efetuar o retrocesso quando um cliente está a comprar um bilhete. Ou seja, quando é selecionado o percurso, caso o cliente queira voltar atrás, tem todas as definições exatamente como as deixou quando decidiu confirmar o percurso selecionado.

ReFactoring

Bad Smells encontrados:	Número de ocasiões repetidas
Long Method:	5
Message Chain	2
Duplicated Code	1

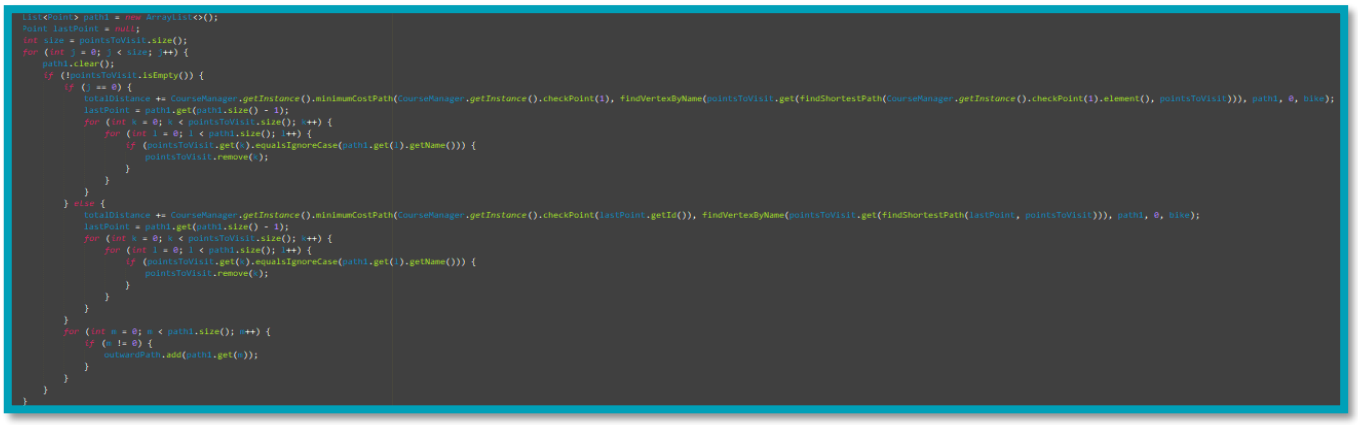
LONG METHOD

```
btnAdicionar = createButton("Adicionar");
btnAdicionar.setOnAction(e -> {
    if (!listSelecionados.getItems().contains(listDisponiveis.getSelectionModel().getSelectedItem())) {
        if (bike == true) {
            String name = listDisponiveis.getSelectionModel().getSelectedItem();
            Vertex<Point> pointToAdd = findVertexByName(name);

            boolean canCross = false;
            for (Edge<Conection, Point> edge : CourseManager.getInstance().getGraph().incidentEdges(pointToAdd)) {
                if (edge.element().isNavigability()) {
                    canCross = true;
                }
            }

            if (canCross == true) {
                listSelecionados.getItems().add(listDisponiveis.getSelectionModel().getSelectedItem());
                listDisponiveis.getItems().remove(listDisponiveis.getSelectionModel().getSelectedItem());

                List<Point> aux1 = new ArrayList<>();
                List<Point> aux2 = new ArrayList<>();
                textDistancia.setText(Integer.toString(distance(aux1, aux2)));

                List<Point> aux3 = new ArrayList<>();
                List<Point> aux4 = new ArrayList<>();
                textPreco.setText(Integer.toString(cost(aux3, aux4)));
            } else {
                popUpErrorPath();
            }
        } else {
            listSelecionados.getItems().add(listDisponiveis.getSelectionModel().getSelectedItem());
            listDisponiveis.getItems().remove(listDisponiveis.getSelectionModel().getSelectedItem());

            List<Point> aux1 = new ArrayList<>();
            List<Point> aux2 = new ArrayList<>();
            textDistancia.setText(Integer.toString(distance(aux1, aux2)));

            List<Point> aux3 = new ArrayList<>();
            List<Point> aux4 = new ArrayList<>();
            textPreco.setText(Integer.toString(cost(aux3, aux4)));
        }
    }
});
```

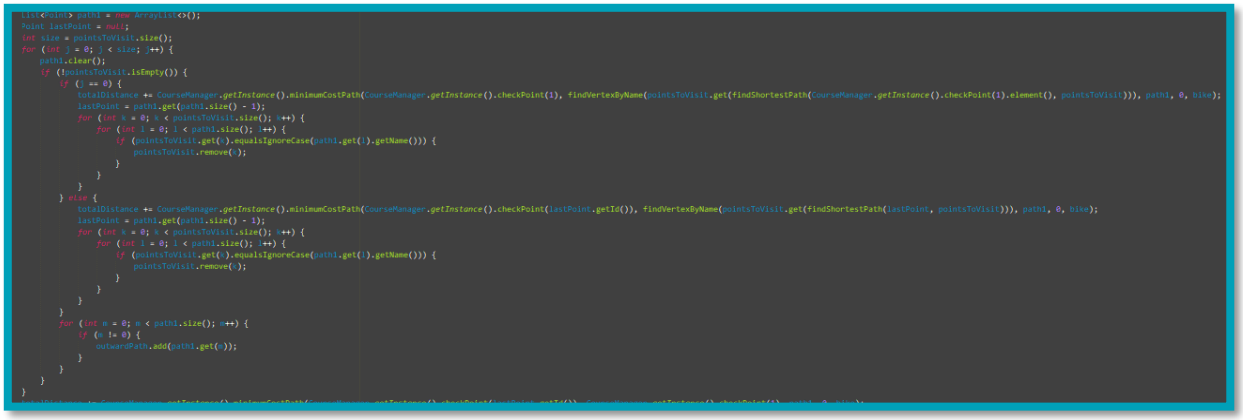
De modo a solucionar este problema, em todas as repetições que o mesmo aconteceu, foram criados métodos adicionais (Extract Method) que executam a ação necessária, sendo então apenas executada a chamada desses métodos em cada situação.

```
btnAdicionar = createButton("Adicionar");
btnAdicionar.setOnAction(e -> {
    addPath();
});
```

MESSAGE CHAIN



De modo a solucionar este problema, em todas as repetições que o mesmo aconteceu, foi utilizada a solução Hide Delegate.



DUPLICATED CODE

```
if (listSelecionados.getItems().size() < 1) {
    popUpErrorPoints();
} else if (radioDistancia.isSelected() && (listSelecionados.getItems().size()) >= 1) {
    //Creates ticket with the path the user wants to visit using the distance
    int criteria = 0;
    Ticket ticket = new Ticket(stage, finalDistancePath1, finalDistancePath2, bike, criteria, caretaker);
    Scene scene = new Scene(ticket, ticket.getWidth(), ticket.getHeight());
    stage.setScene(scene);
    stage.centerOnScreen();
    stage.show();
} else if (radioPreco.isSelected() && (listSelecionados.getItems().size()) >= 1) {
    //Creates ticket with the path the user wants to visit using the cost
    int criteria = 1;
    Ticket ticket = new Ticket(stage, finalCostPath1, finalCostPath2, bike, criteria, caretaker);
    Scene scene = new Scene(ticket, ticket.getWidth(), ticket.getHeight());
    stage.setScene(scene);
    stage.centerOnScreen();
    stage.show();
}
```

Através da solução Extract Method, foi efetuado um método auxiliar que efetua a mesma ação do código inicial.

```
    if (!checkAmountOfPoints()) {
        popUpErrorPoints();
    } else if (radioDistancia.isSelected() && checkAmountOfPoints()) {
        //Creates ticket with the path the user wants to visit using the distance
        createTicketScreen(finalDistancePath1, finalDistancePath2, bike, 0, caretaker);
    } else if (radioPreco.isSelected() && checkAmountOfPoints()) {
        //Creates ticket with the path the user wants to visit using the cost
        createTicketScreen(finalCostPath1, finalCostPath2, bike, 1, caretaker);
    }
}

private void createTicketScreen(List<Point> list1, List<Point> list2, boolean bike, int criteria, Caretaker caretaker) {
    Ticket ticket = new Ticket(stage, list1, list2, bike, criteria, caretaker);
    Scene scene = new Scene(ticket, ticket.getWidth(), ticket.getHeight());
    stage.setScene(scene);
    stage.centerOnScreen();
    stage.show();
}
```