

# Programação Avançada 2018/2019

Licenciatura em Eng<sup>a</sup>. 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
TADs implementadas .....	4
Diagrama de classes.....	5
Javadoc .....	9
Padrões de software .....	10
Refactoring .....	12
Libraries/JarFiles utilizados .....	28
Tecnologias utilizadas .....	29

## 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 seleccionado 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 seleccionados 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

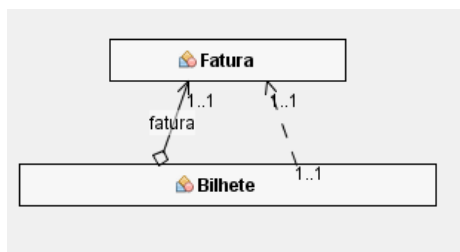


Fig.1

### 2- DAO

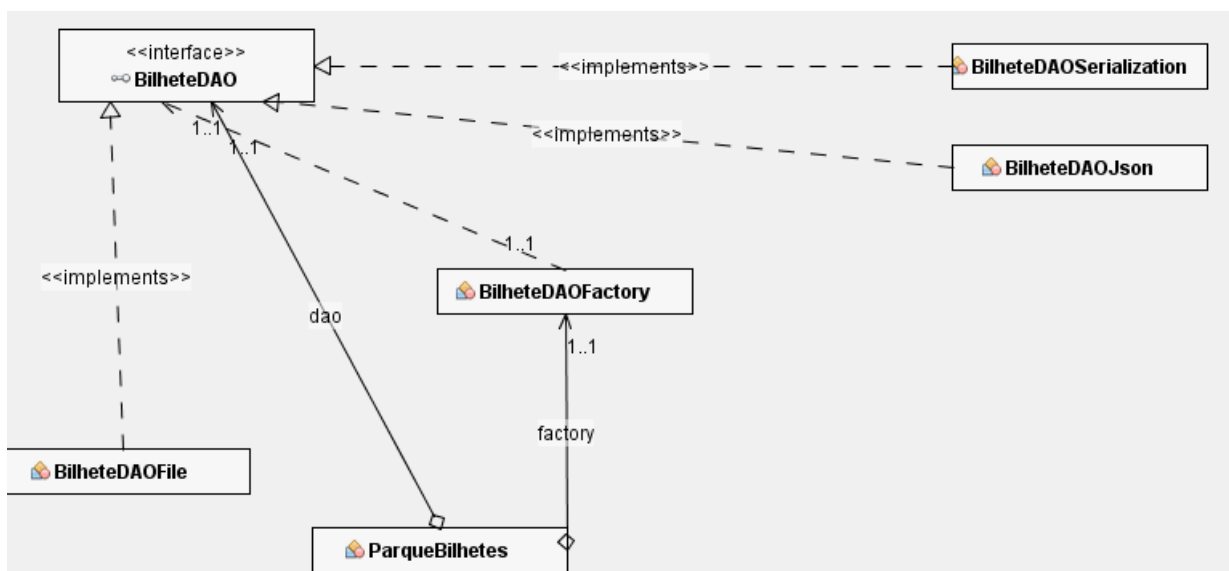


Fig.2

### 3- MVC CONTROLLER



Fig.3

4- MVC MODELS

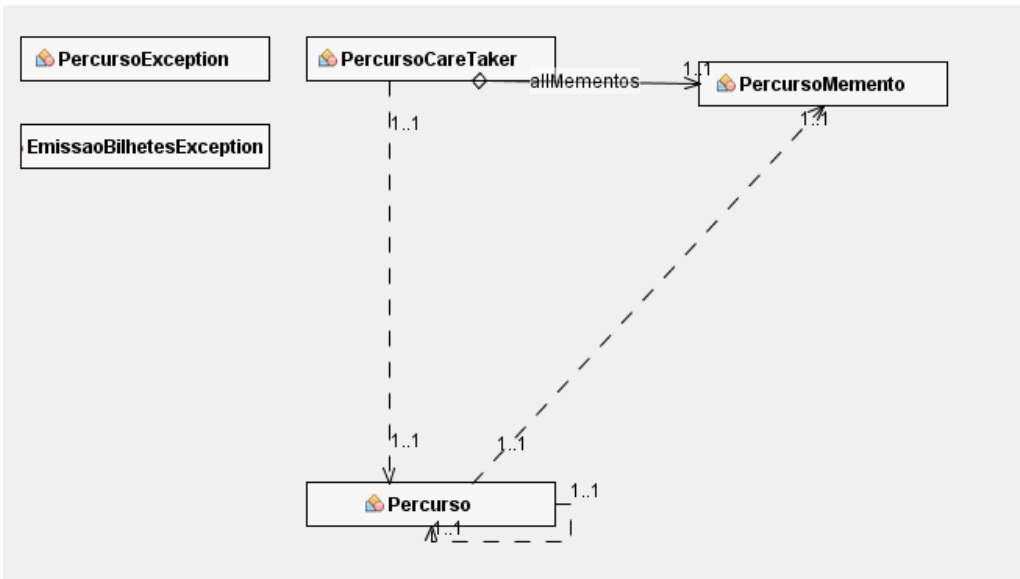


Fig.4

5- MVC VIEWS

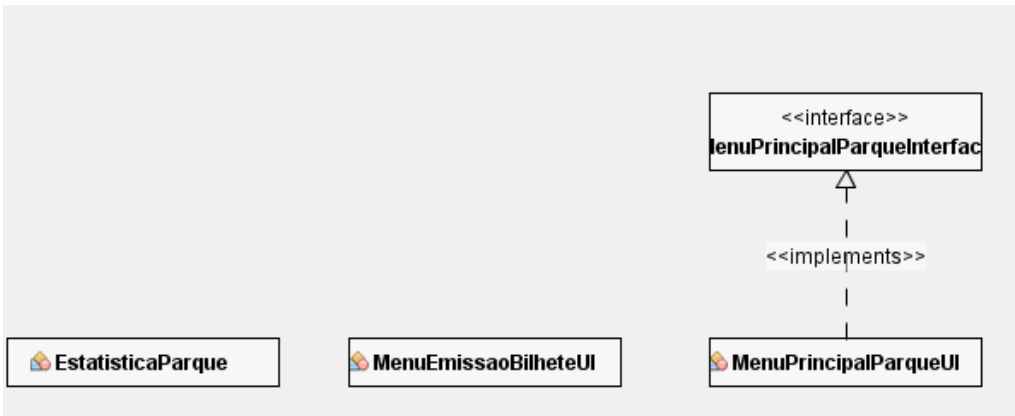


Fig.5

6- PDFGenerator



Fig.6

## 7- SINGLETON

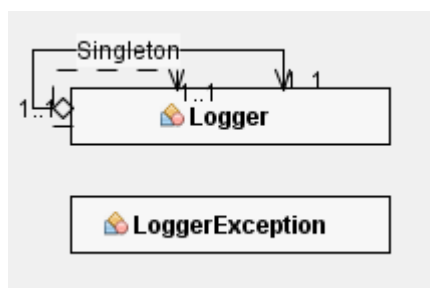


Fig.7

## 8- MAIN

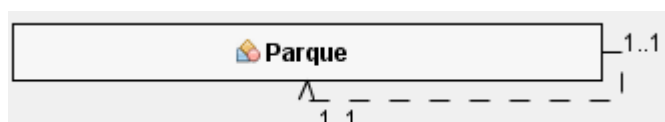


Fig.8

## 9- DIGRAPH GRAPH

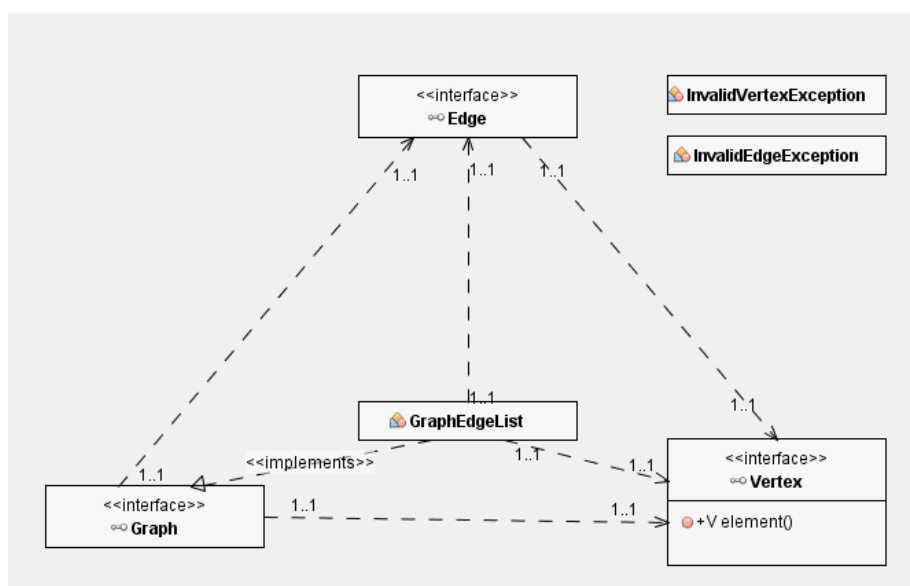


Fig.9

10- DIGRAPH MODEL

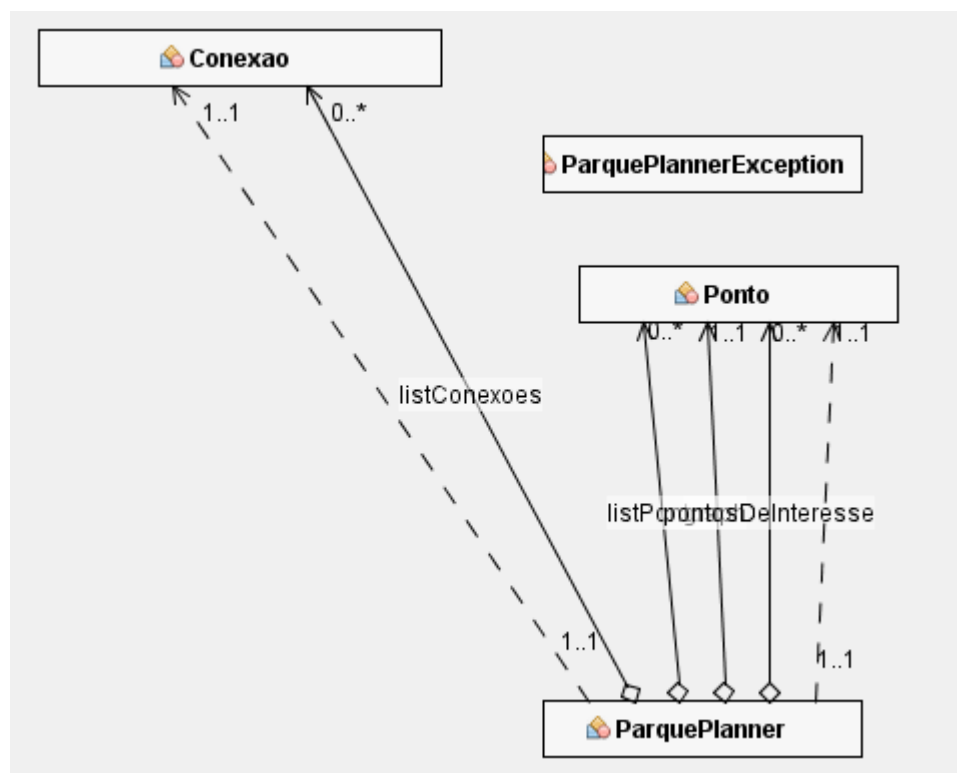


Fig.10



## 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.

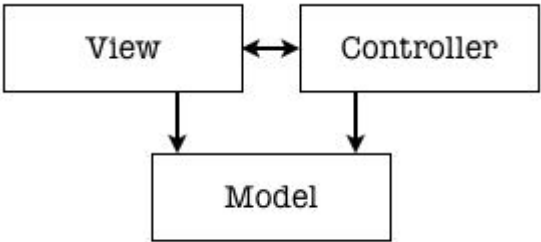
**GenerateTicket** - Classe que gera os PDF's de fatura e bilhete.

**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 Access Object (DAO)	<p>DAO é um padrão de projetos onde um objeto:</p> <ul style="list-style-type: none"> <li>• provê uma interface que abstrai o acesso a dados;</li> <li>• lê e grava a partir da origem de dados (banco de dados, arquivo, memória, etc.);</li> <li>• encapsula o acesso aos dados, de forma que as demais classes não precisam saber sobre isso.</li> </ul>	<p>Padrao que guarda o Bilhete como objecto, e todos os seus atributos.</p> <p>Tipos Implementados:</p> <ol style="list-style-type: none"> <li>1. BilheteDAOFile</li> <li>2. BilheteDAOJson</li> <li>3. BilheteDAOSerialization</li> </ol>
Model View Controller (MVC)	<p>Padrão de arquitetura de software que separa a representação da informação da interação do usuário com ele. Este padrão separa componentes maiores possibilitando a reutilização de código e desenvolvimento paralelo de maneira eficiente.</p>	 <pre> graph TD     View[View] &lt;--&gt; Controller[Controller]     View --&gt; Model[Model]     Controller --&gt; Model     </pre> <p>Componentes Principais</p> <p>Models:</p> <ul style="list-style-type: none"> <li>• Percurso</li> </ul> <p>Views:</p> <ul style="list-style-type: none"> <li>• MenuPrincipalParqueUI</li> <li>• MenuEmissaoBilheteUI</li> </ul> <p>Controller:</p> <ul style="list-style-type: none"> <li>• PercursoController</li> </ul> <p>MVC é uma aquitectura 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.</p>
Observer	<p>Padrão de projeto de software que define uma dependência um-para-muitos entre objetos de modo a que quando um objeto muda o estado, todos os seus dependentes são notificados e atualizados automaticamente.</p>	<p>Observable: ParquePlanner</p> <p>Observer: MenuPrincipalParqueInterface</p> <pre> @Override public void update(java.util.Observable o, Object arg)     </pre> <p>O modelo parquePlanner é o modelo que está a ser observado e manipulado pela view que é o observador</p>

Singleton	Padrão de projeto de software que garante a existência de apenas uma instância de uma classe, mantendo um ponto global de acesso ao seu objeto.	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 determinado 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	<pre> @Override public void start(Stage primaryStage) throws FileNotFoundException, InterruptedException {     ParquePlanner planner = new ParquePlanner();     planner.readFile(planner); }  public void readFile(ParquePlanner planner) {     Scanner scan = new Scanner(System.in);     File f = new File("src\\parque\\mapa0.dat");     System.out.println("-&gt;Enter name of file here : ");     String nameFile;     nameFile = scan.nextLine();     scan.close();     System.out.println(nameFile);      try {         FileReader filereader = new FileReader(f);          BufferedReader bufferedReader = new BufferedReader(filereader);          while ((line = bufferedReader.readLine()) != null) {             String[] token = line.split(" ");              if (token.length == 2) {                 Ponto ponto = new Ponto(token[0], token[1]);                 planner.addPonto(ponto);                 System.out.println(ponto);             } else if (token.length &gt; 2) {                  Conexao conexao = new Conexao(Integer.parseInt(token[0]), Enum.valueOf(TYPE_CONECT.class, token[1]), token[2], token[3], token[4]);                 planner.addConexao(planner.getPontoById(token[2]), planner.getPontoById(token[3]), conexao);             }             System.out.println("Sucesso na Leitura");         }     } catch (FileNotFoundException e) {     } } </pre>
Após refactoring	<pre> @Override public void start(Stage primaryStage) throws FileNotFoundException, InterruptedException {     ParquePlanner planner = new ParquePlanner(); }  public ParquePlanner() {     this.graph = new GraphEdgeList&lt;&gt;();     this.listPontos = new ArrayList&lt;&gt;();     this.listConexoes = new ArrayList&lt;&gt;();      caretaker = new PercursoCareTaker();     this.pontosDeInteresse = new ArrayList&lt;&gt;();      readFile(); }  private void readFile() { </pre>

## Aplicação para Emissão de Bilhete - Percurso num Parque Biológico

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 mensagem do erro.
Técnica de refactoring	Introduce Assertion- Peçaço de código assume alguma coisa sobre o estado do programa. Fazer com que a assunção seja explicita consoante a asserção.
Antes refactoring	<pre> private void readFile() {     Scanner scan = new Scanner(System.in);     File f = new File("src\\parque\\mapa0.dat");     System.out.println("-&gt;Enter name of file here : ");     String nameFile;     nameFile = scan.nextLine();     scan.close();     System.out.println(nameFile);     if (nameFile.equals("mapa0.dat")) {         try {              FileReader filereader = new FileReader(f);             BufferedReader bufferReader = new BufferedReader(filereader);             while ((line = bufferReader.readLine()) != null) {                 String[] token = line.split(", ");                 if (token.length == 2) {                     Ponto ponto = new Ponto(token[0], token[1]);                     this.addPonto(ponto);                     System.out.println(ponto);                 } else if (token.length &gt; 2) {                      Conexao conexao = new Conexao(Integer.parseInt(token[0]), Enum.valueOf(TYPE_CONECT.class, token[1]).toString(), token[2], token[3], token[4]);                     this.addConexao(this.getPontoById(token[3]), this.getPontoById(token[4]), conexao);                 }                 System.out.println("Sucesso na Leitura");             }         } catch (FileNotFoundException e1) {             e1.             System.out.println("File not Found #404!");         } catch (IOException e) {         }     } } </pre>
Após refactoring	<pre> private void readFile() {     File f;     String nameFile;     try (Scanner scan = new Scanner(System.in)) {         f = new File("src\\parque\\mapa0.dat");         System.out.println("-&gt;Enter name of file here : ");         nameFile = scan.nextLine();     }     System.out.println(nameFile);     if (nameFile.equals("mapa0.dat")) {         try {              FileReader filereader = new FileReader(f);             BufferedReader bufferReader = new BufferedReader(filereader);             while ((line = bufferReader.readLine()) != null) {                 String[] token = line.split(", ");                 if (token.length == 2) {                     Ponto ponto = new Ponto(token[0], token[1]);                     this.addPonto(ponto);                     System.out.println("Ponto Lido " + ponto.getId());                 } else if (token.length &gt; 2) {                     Conexao conexao = new Conexao(Integer.parseInt(token[0]), Enum.valueOf(TYPE_CONECT.class, token[1].toUpperCase()), token[2], token[3], token[4]);                     this.addConexao(this.getPontoById(token[3]), this.getPontoById(token[4]), conexao);                     System.out.println("Conexao Lida " + conexao.getId());                 }             }         } catch (FileNotFoundException e1) {             System.out.println("Exception thrown : " + e1);             System.out.println("FileNot Found");         } catch (IOException e) {             System.out.println("Exception thrown : " + e);         }     } } </pre>

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	<pre> public void actionCalcular() throws PercursoException {     List&lt;Conexao&gt; mapa = new ArrayList&lt;&gt;();     if (view.getToggleGroupLomotion().getSelectedToggle().getUserData() != null) {         if ((ParquePlanner.Criterias) view.getToggleGroupCriteria().getSelectedToggle().getUserData() != null) {             if (!parque.getPontosDeInteresse().isEmpty()) {                 if (parque.gerarPercurso((ParquePlanner.TipoPercurso) view.getToggleGroupLomotion().getSelectedToggle().getUserData(),                     (ParquePlanner.Criterias) view.getToggleGroupCriteria().getSelectedToggle().getUserData(),                     parque.getPontosDeInteresse(), mapa) != null) {                     Percurso percurso = parque.gerarPercurso((ParquePlanner.TipoPercurso) view.getToggleGroupLomotion().getSelectedToggle().getUs                         (ParquePlanner.Criterias) view.getToggleGroupCriteria().getSelectedToggle().getUserData(),                         parque.getPontosDeInteresse(), mapa);                     System.out.println("Percurso Calculado");                     desenharPath();                     percurso.setCalculado(true);                     String msg = "";                     msg += " Custo:" + percurso.getCusto() + " Distancia" + percurso.getDistancia();                     view.calcular(msg);                 } else {                     System.out.println("Percurso Não Calculado - why?");                 }             } else {                 System.out.println("Percurso Não Calculado -Pontos Interesse");             }         } else {             System.out.println("Percurso Não Calculado Tipo de Criterias");         }     } else {         view.showError("Locomoção");     } } </pre>
Após refactoring	<pre> public void actionCalcular() throws PercursoException {      try {         List&lt;Conexao&gt; mapa = new ArrayList&lt;&gt;();         Percurso percurso = parque.gerarPercurso(             (TipoPercurso) view.getToggleGroupLomotion().getSelectedToggle().getUserData(),             (Criterias) view.getToggleGroupCriteria().getSelectedToggle().getUserData(),             parque.getPontosDeInteresse(),             mapa);          view.calcular(percurso.getCusto(), percurso.getDistancia());         percurso.setCalculado(true);         desenharPath();      } catch (PercursoException e) {          view.showError(e.getMessage());     } } </pre>

Bad smell	
Técnica de refactoring	Replace Error Code with Exception
Antes refactoring	<pre> @Override public ToggleGroup getToggleGroupLomotion() {      return groupLocomotion; } </pre>
Após refactoring	<pre> @Override public ToggleGroup getToggleGroupLomotion() {      if (this.groupLocomotion.getSelectedToggle() == null) {         throw new PercursoException("Selecione um tipo de locomoção");     } else {         return groupLocomotion;     } }  public void actionAddPonto() {     try {         view.getToggleGroupLomotion();     } } </pre>



Bad smell	
Técnica de refactoring	Replace Error Code with Exception
Antes refactoring	<pre> @Override public ToggleGroup getToggleGroupCriteria() {     return groupCriteria; } </pre>
Após refactoring	<pre> @Override public ToggleGroup getToggleGroupCriteria() {     if (this.groupCriteria.getSelectedToggle() == null) {         throw new PercursoException("Selecione um tipo de Criterio de Calculo");     } else {         return groupCriteria;     } }  public void actionAddPonto() {     try {         view.getToggleGroupLomotion();         view.getToggleGroupCriteria();     } } </pre>

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	<pre> @Override public Button getBtEmitirBilhete() {      return btEmitirBilhete; }  public void actionEmitirBilhete() throws PercursoException {     try {         if (parque.getPercursoActual().isCalculado()) {             Tab t3 = this.tabpane.getTabs().get(2);             Tab t = this.tabpane.getTabs().get(0);             t.setContent(new MenuEmissaoBilheteUI(parque, model));         }     } catch (PercursoException e) {         view.showError(e.getMessage());     } } </pre>
Após refactoring	<pre> @Override public Button getBtEmitirBilhete() {     if (parque.getPercursoActual() == null    parque.getPercursoActual().isCalculado() == false) {         throw new PercursoException("Impossível Emitir Bilhete, Percurso não foi calculado");     } else {         return btEmitirBilhete;     } }  public void actionEmitirBilhete() throws PercursoException {     try {         view.getBtEmitirBilhete();         Tab tabActual = this.tabpane.getTabs().get(0);         tabActual.setContent(new MenuEmissaoBilheteUI(parque, model));     } catch (PercursoException e) {         view.showError(e.getMessage());     } } </pre>

Bad smell	
Técnica de refactoring	Move Method
Explicação	Removemos o ciclo <i>for</i> dentro do constructor e criamos o metodo que é responsável por adicionar pontos à comboBox.
Antes refactoring	<pre>     */     public PercursoController(PercursoManager model, MenuPrincipalParqueInterface view, P         this.model = model;         this.view = view;         this.parque = parque;         this.graphView = graphView;         this.tabpane = tabpane;         for (Ponto item : parque.getListPontos()) {             view.getPontoComboBox().getItems().add(item.getId());         }         view.setTriggers(this);         model.addObserver(view);         this.parque.addObserver(view);     } </pre>
Após refactoring	<pre>     public PercursoController(PercursoManager model, MenuPrincipalParqueInterface view,         this.model = model;         this.view = view;         this.parque = parque;         this.graphView = graphView;         this.tabpane = tabpane;         initiateView(view);         model.addObserver(view);         this.parque.addObserver(view);     }      private void initiateView(MenuPrincipalParqueInterface view) {         parque.getListPontos().forEach((item) -&gt; {             view.getPontoComboBox().getItems().add(item.getId());         });         view.setTriggers(this);     } </pre>

Bad smell	Dead Code
Explicação	PercursoManager <i>class</i> 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	<pre> /**  * Constructor method of PercursoManager  *  * @param name  */ public PercursoManager(String name) {     Percurso = new Percurso(name);     caretaker = new PercursoCareTaker(); }  /**  * Getter method for Percurso PontosCaminho  *  * @return PontosCaminho of Percurso  */ public List&lt;Ponto&gt; getPontosCaminho() {     return Percurso.getPontosCaminho(); } </pre>
Após refactoring	Eliminada a classe inutilizada.

Bad smell	Duplicated code
Explicação	Definir as condições no metodo getNifInput() para capturar e tratar os erros na acção do generateBilhete()
Técnica de refactoring	Extract Method-Mover o código para um novo metodo separado e substituir o código por uma chamada para o novo metodo
Antes refactoring	<pre> public TextField getNifInput() {     return nifInput; }  /**  * Setter method for the triggers  */ public void setTriggers() {     GenerateBilhete.setOnAction((ActionEvent event) -&gt; {         if (getNifInput().getText().equals("")) {             //Erro             Fatura fatura = new Fatura("999999999");              Bilhete bilhete = new Bilhete("teste", parque.getPercursoActual(), fatura);              d.generateBilhetePdf(bilhete);             this.parqueBilhetes.insert(bilhete);          } else if (getNifInput().getText().length() == 9) {             Fatura fatura = new Fatura(getNifInput().getText());             Bilhete bilhete = new Bilhete("teste", parque.getPercursoActual(), fatura);             d.generateBilhetePdf(bilhete);             d.generateFaturaPdf(fatura);             this.parqueBilhetes.insert(bilhete);          } else {              this.lblError.setText("Insira um Valor Válido");          }      }); } </pre>
Após refactoring	<pre> public TextField getNifInput() {     if (nifInput.getText().length() &lt; 9 &amp;&amp; nifInput.getText().length() &gt;= 1) {         throw new EmissaoBilhetesException("Insira um Valor Válido");     } else if (nifInput.getText().equals("")) {         this.nifInput.setText("999999999");         return nifInput;     } else {         return nifInput;     } }  public void clear() {     lblError.setText("");     lblResult.setText(""); }  public void setTriggers() {     GenerateBilhete.setOnAction((ActionEvent event) -&gt; {         try {             clear();             Fatura fatura = new Fatura(getNifInput().getText());             Bilhete bilhete = new Bilhete("ParqueBioTicket", parque.getPercursoActual(), fatura);             d.generateBilhetePdf(bilhete);             this.parqueBilhetes.insert(bilhete);             this.lblResult.setText("Bilhete Impresso");         } catch (EmissaoBilhetesException e) {             this.lblError.setText(e.getMessage());         }      }); } </pre>

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	<pre> @Override public void start(Stage primaryStage) throws FileNotFoundException, InterruptedException {     ParquePlanner planner = new ParquePlanner();     TabPane tabpane = new TabPane();     tabpane.setTabClosingPolicy(TabClosingPolicy.UNAVAILABLE);     SingleSelectionModel&lt;Tab&gt; selectionModel = tabpane.getSelectionModel();     Tab tab1 = new Tab();     Tab tab3 = new Tab();     tab3.setText("ESTATISTICAS");     EstatisticaParque t = new EstatisticaParque();     tab3.setContent(t);     tab1.setText("MENU");     Tab tab2 = new Tab();     tab2.setText("MAPA");     VertexPlacementStrategy strategy = new CircularSortedPlacementStrategy();     GraphPanel&lt;Ponto, Conexao&gt; graphView = new GraphPanel&lt;&gt;(planner.getGraph(), strategy);     // PercursoManager model = new PercursoManager("Menu Parque");     MenuPrincipalParqueUI view = new MenuPrincipalParqueUI(planner);     PercursoController controller = new PercursoController( view, planner, graphView, tabpane);     Pane window = new Pane();     tab1.setContent(window);     tab2.setContent(graphView);     tabpane.getStylesheets().addAll(this.getClass().getResource("/javafxgraphs/ui/resources/TabPaneStyle.css").toExternalForm());     tabpane.getTabs().add(tab1);     tabpane.getTabs().add(tab2);     tabpane.getTabs().add(tab3);     window.getChildren().add(view);     Scene sceneTeste = new Scene(tabpane, 1500, 600);     primaryStage.setTitle("Parque Bio");     primaryStage.setScene(sceneTeste);     primaryStage.setOpacity(0.9);     primaryStage.show();     graphView.plotGraph(); </pre>
Após refactoring	<pre> private TabPane configTabPane(MenuPrincipalParqueUI view, GraphPanel graphView) {     TabPane tabpane = new TabPane();     Tab tab1 = new Tab("MENU", view);     Tab tab2 = new Tab("MAPA", graphView);     Tab tab3 = new Tab("ESTATISTICAS", new EstatisticaParque());     tabpane.getTabs().addAll(tab1, tab2, tab3);     tabpane.getStylesheets().addAll(this.getClass().getResource("/javafxgraphs/ui/resources/     tabpane.setTabClosingPolicy(TabClosingPolicy.UNAVAILABLE);     return tabpane; }  @Override public void start(Stage primaryStage) throws FileNotFoundException, InterruptedException {     //MODEL     ParquePlanner planner = new ParquePlanner();     //GRAPH     VertexPlacementStrategy strategy = new CircularSortedPlacementStrategy();     GraphPanel&lt;Ponto, Conexao&gt; graphView = new GraphPanel&lt;&gt;(planner.getGraph(), strategy);     //VIEW     MenuPrincipalParqueUI view = new MenuPrincipalParqueUI(planner);     //TABPANE     TabPane tabpane = configTabPane(view, graphView);     //CONTROLLER     PercursoController controller = new PercursoController(view, planner, graphView, tabpane);      Scene sceneTeste = new Scene(tabpane, 1259, 606);     primaryStage.setTitle("Parque Bio");     primaryStage.setScene(sceneTeste);     primaryStage.setOpacity(0.9);     primaryStage.show();     graphView.plotGraph();      setPathsColor(planner, graphView); </pre>

Bad smell	Long Method
Explicação	O metodo estava confuso e demasiado longo logo fragmentamos o codigo e fizemos uma chamada aos novos metodos( <u><i>setPathsColor(planner, graphView);</i></u> <u><i>setArrows(graphView);</i></u> ) no <i>start()</i>
Técnica de refactorin g	Extract Method- Mover o codigo para um novo metodo separado e substituir o código por uma chamada para o novo metodo
Antes refactorin g	<pre> for (Edge&lt;Conexao, Ponto&gt; item : planner.getGraph().edges()) {     if (item.element().getTipo().equals(TYPE_CONNECT.PONTE)) {         graphView.setEdgeColor(item, Color.DODGERBLUE, 0.8);      } else {          graphView.setEdgeColor(item, Color.CYAN, 0.5);      } }  for (Map.Entry&lt;Edge&lt;Conexao, Ponto&gt;, GraphEdge&gt; i : graphView.getGraphEdgeMap().entrySet()) {     if (i.getKey().element().getTipo().equals(TYPE_CONNECT.CAMINHO)) {         double[] arrowShape = new double[]{0, 0, GRAPH_EDGE_WIDTH, 2 * GRAPH_EDGE_WIDTH, -GRAPH_EDGE_WIDTH, 2 * G          Arrow arrow1 = new Arrow(i.getValue(), ARROW_COLOR, 0.2f, arrowShape);         i.getValue().addArrow(arrow1);         graphView.getChildren().add(arrow1);     } else {      } } </pre>
Após refactorin g	<pre> setPathsColor(planner, graphView); setArrows(graphView);  }  private void setPathsColor(ParquePlanner planner, GraphPanel&lt;Ponto, Conexao&gt; graphView) {     for (Edge&lt;Conexao, Ponto&gt; item : planner.getGraph().edges()) {         if (item.element().getTipo().equals(TYPE_CONNECT.PONTE)) {             graphView.setEdgeColor(item, Color.DODGERBLUE, 0.8);         } else {             graphView.setEdgeColor(item, Color.CYAN, 0.5);         }     } }  private void setArrows(GraphPanel&lt;Ponto, Conexao&gt; graphView) {     for (Map.Entry&lt;Edge&lt;Conexao, Ponto&gt;, GraphEdge&gt; item : graphView.getGraphEdgeMap().entrySet()) {         if (item.getKey().element().getTipo().equals(TYPE_CONNECT.CAMINHO)) {             double[] arrowShape = new double[]{0, 0, GRAPH_EDGE_WIDTH, 2 * GRAPH_EDGE_WIDTH, -GRAPH_EDGE_WIDTH, 2 * G              Arrow arrow1 = new Arrow(item.getValue(), ARROW_COLOR, 0.2f, arrowShape);             item.getValue().addArrow(arrow1);             graphView.getChildren().add(arrow1);         }     } } </pre>

Bad smell	Dead Code
Explicação	Excluir o código não utilizado
Técnica de refactoring	Método que não é mais utilizado pois tornou-se obsoleto.
Antes refactoring	<pre>public void generatePdfTicket () { }</pre>
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	<pre>public class Menu extends BorderPane {      /**      * Constructor method for class Menu      * @param planner      * @throws FileNotFoundException      */     public Menu(ParquePlanner planner) throws FileNotFoundException {         window(planner);     } }</pre>
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	<pre> private VBox criação() {     int totalPe = 0;     int totalBike = 0;     int todos = 0;     List&lt;Bilhete&gt; lista = new ArrayList&lt;&gt;();     lista = b.selectAll(); // lista = database.getAll();     List&lt;Bilhete&gt; listaPe = new ArrayList&lt;&gt;();     List&lt;Bilhete&gt; listaBike = new ArrayList&lt;&gt;();     for (Bilhete item : lista) {         if (item.getPercurso().getTipoPercurso().equals(TipoPercurso.PE)) {             listaPe.add(item);         } else {             listaBike.add(item);         }     }     totalPe = listaPe.size();     totalBike = listaBike.size();     todos = lista.size();     int percentagemPe = 0;     int percentagemBike = 0;      percentagemPe = (totalPe * 100) / todos;     percentagemBike = (totalBike * 100) / todos;     ObservableList&lt;PieChart.Data&gt; pieChartData =         FXCollections.observableArrayList(             new PieChart.Data("A pé: " + String.valueOf(totalPe) + "   " + String.valueOf((totalPe / todos) * 100) + "%", percentagemPe),             new PieChart.Data("Bicicleta: " + String.valueOf(totalBike) + "   " + String.valueOf((totalBike / todos) * 100) + "%", percentagemBike));     final PieChart chart = new PieChart(pieChartData);     pieChartData.stream().forEach(pieData -&gt; {         pieData.getNode().addEventHandler(MouseEvent.MOUSE_CLICKED, event -&gt; {             Bounds bl = pieData.getNode().getBoundsInLocal();             double newX = (bl.getWidth() / 2 + bl.getMinX());             double newY = (bl.getHeight() / 2 + bl.getMinY());             // Make sure pie wedge location is reset             pieData.getNode().setTranslateX(0);             pieData.getNode().setTranslateY(0);             TranslateTransition tx = new TranslateTransition(                 Duration.millis(1500), pieData.getNode());             tx.setByX(newX);             tx.setByY(newY);             tx.setAutoReverse(true);             tx.setCycleCount(2);             tx.play();});});     VBox vbox = new VBox();     vbox.getChildren().addAll(chart);     return vbox; } </pre>
Após refactoring	<pre> private VBox criterio() {     List&lt;Bilhete&gt; lista = b.selectAll();     // lista = database.getAll();     List&lt;Bilhete&gt; listaPe = new ArrayList&lt;&gt;();     List&lt;Bilhete&gt; listaBike = new ArrayList&lt;&gt;();     for (Bilhete item : lista) {         if (item.getPercurso().getTipoPercurso().equals(TipoPercurso.PE)) {             listaPe.add(item);         } else {             listaBike.add(item);         }     }     totalPe = listaPe.size();     totalBike = listaBike.size();     todos = lista.size();      percentagemPe = (totalPe * 100) / todos;     percentagemBike = (totalBike * 100) / todos;     pieChartData = FXCollections.observableArrayList(         new PieChart.Data("A pé: " + String.valueOf(totalPe) + "   " + percentagemPe + "%", percentagemPe),         new PieChart.Data("Bicicleta: " + String.valueOf(totalBike) + "   " + percentagemBike + "%", percentagemBike));     chartPercentagem = new PieChart(pieChartData);     defineEffect(chartPercentagem);     VBox vbox = new VBox();     vbox.getChildren().addAll(chartPercentagem);     return vbox; } </pre>

*Aplicação para Emissão de Bilhete - Percurso num Parque Biológico*

<b>Bad smell</b>	Dead Code-Unused Parameter
<b>Explicação</b>	Reparámos que existiam vários imports desnecessários e inutilizados
<b>Técnica de refactoring</b>	Remove Parameter
<b>Antes refactoring</b>	<pre>import dijkstra.graph.Edge; import dijkstra.graph.Vertice; import dijkstra.model.Ti; import dijkstra.model.Conexao; import dijkstra.model.Conexao.TYPE_CONECT; import dijkstra.model.ParquePlanner; import dijkstra.model.ParquePlanner.Criterias; import dijkstra.model.Ponto; import java.io.BufferedReader; import java.io.File; import java.io.FileInputStream; import java.io.FileNotFoundException; import java.io.FileReader; import java.io.IOException; import java.net.URL; import java.util.ArrayList; import java.util.LinkedList; import java.util.List; import java.util.Map; import java.util.HashMap; import java.util.Scanner; import java.util.Set; import java.util.concurrent.TimeUnit; import javafx.animation.FadeTransition; import javafx.application.Application; import javafx.scene.Group; import javafx.scene.Scene; import javafx.scene.control.Button; import javafx.scene.control.DatePicker; import javafx.scene.control.ToolBar; import javafx.scene.image.Image; import javafx.scene.image.ImageView; import javafx.scene.layout.BorderPane; import javafx.scene.paint.Color; import javafx.stage.Stage; import javafx.stage.StageStyle; import javafx.scene.v2.graphview.CircularSortedPlacementStrategy; import javafx.scene.v2.graphview.GraphPanel; import javafx.scene.v2.graphview.VertexPlacementStrategy; import javafx.scene.control.ProgressBar; import javafx.scene.control.SingleSelectionModel; import javafx.scene.control.Tab; import javafx.scene.control.TabPane; import javafx.scene.control.TabPane.TabClosingPolicy; import javafx.scene.effect.Blend; import javafx.scene.effect.DropShadow; import javafx.scene.effect.GaussianBlur; import javafx.scene.effect.Slide; import javafx.scene.effect.Light; import javafx.scene.effect.Lighting; import javafx.scene.effect.PerspectiveTransform; import static javafx.scene.input.KeyCode.R; import javafx.scene.layout.Pane; import static javafx.scene.paint.Color.BLACK; import javafx.util.Duration; import javafx.scene.v2.graphview.Arrow; import javafx.scene.v2.graphview.GraphEdge; import static javafx.scene.v2.graphview.GraphPanel.ARROW_COLOR; import static javafx.scene.v2.graphview.GraphPanel.EDGE_WIDTH; import javafx.scene.v2.graphview.RoundPlacementStrategy;</pre>
<b>Após Refactoring</b>	<pre>import MVC.CONTROLLERS.PercursosController; import MVC.VIEWS.EstatisticaParque; import MVC.VIEWS.MenuPrincipalParqueUI; import dijkstra.graph.Edge; import dijkstra.model.Conexao; import dijkstra.model.Conexao.TYPE_CONECT; import dijkstra.model.ParquePlanner; import dijkstra.model.Ponto; import java.io.FileNotFoundExcepção; import java.util.Map; import javafx.application.Application; import javafx.scene.Scene; import javafx.scene.paint.Color; import javafx.stage.Stage; import javafx.scene.v2.graphview.CircularSortedPlacementStrategy; import javafxscene_v2.graphview.GraphPanel; import javafxscene_v2.graphview.VertexPlacementStrategy; import javafx.scene.control.Tab; import javafx.scene.control.TabPane; import javafx.scene.control.TabPane.TabClosingPolicy; import javafxscene_v2.graphview.Arrow; import javafxscene_v2.graphview.GraphEdge; import static javafxscene_v2.graphview.GraphPanel.ARROW_COLOR; import static javafxscene_v2.graphview.GraphPanel.EDGE_WIDTH;</pre>

## Aplicação para Emissão de Bilhete - Percurso num Parque Biológico

Bad smell	Duplicated Code
Explicação	
Técnica de refactoring	
Antes refactoring	<pre> public void generateBilhetePdf(Bilhete bilhete) {     try {         Document document = new Document();         PdfWriter pdfWriter = PdfWriter.getInstance(document, new FileOutputStream("Bilhete.pdf"));         document.open();         Image qrCodeImage = generateQRCodeTicket(bilhete);         qrCodeImage.scalePercent(100);         document.add(qrCodeImage);         document.add(new Paragraph("Bilhete"));         document.add(new Paragraph(bilhete.toString()));         document.close();         generateFaturaPdf(bilhete.getFatura());     } catch (FileNotFoundException e) {         e.printStackTrace();     } catch (DocumentException e) {         e.printStackTrace();     } }  /**  * Method that generates the Fatura into PDF  * @param fatura fatura  */ public void generateFaturaPdf(Fatura fatura) {     try {         Document document = new Document();         PdfWriter pdfWriter = PdfWriter.getInstance(document, new FileOutputStream("Fatura.pdf"));         document.open();         Image qrCodeImage = generateQRCodeFatura(fatura);         qrCodeImage.setAbsolutePosition(20, 500);         qrCodeImage.scalePercent(100);         document.add(qrCodeImage);         document.add(new Paragraph("Fatura"));     } } </pre>
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-3.21.0 - Utilizado para o DAO

## Tecnologias utilizadas

<http://graphonline.ru/en/> - Utilizado para auxílio de cálculo e teste de percursos calculados.