Programação Avançada

2018/2019

Licenciatura em Engª. Informática

Relatório Final

Professor: Bruno Silva

nº180221082 Edgar Santos

nº180221110 Luís Varela

DESCRIÇÃO DO TRABALHO DESENVOLVIDO

INTRODUÇÃO

Este relatório é desenvolvido no âmbito da unidade curricular de Programação Avançada, cujo objetivo deste documento passa por descrever todas as tarefas realizadas ao longo do desenvolvimento do projeto final de Programação Avançada. O projeto consiste no desenvolvimento de uma aplicação de desktop que permite a criação e o cálculo de percursos a pé ou de bicicleta dentro um parque biológico. A aplicação deverá conter informação sobre o preço total a pagar pelo trajeto selecionado e ainda realizar a emissão de bilhetes com a respetiva fatura. O parque biológico é constituído por vários pontos de interesses, que posteriormente serão conectados ou por caminhos, ou pontes. O objetivo deste projeto passa por ajudar o utilizador a calcular percursos de forma a minimizar a distância ou o custo do trajeto.

INTERFACE GRÁFICA

Inicialmente foi feito alguns *mockups* da interface gráfica pretendida para a aplicação, onde teria o ecrã inicial com o mapa do parque e os pontos de interesse que eram contidos nesse mesmo mapa, neste ecrã o utilizador terá acesso a grande parte das funcionalidades da aplicação, também como escolher e compor o trajeto que deseja.

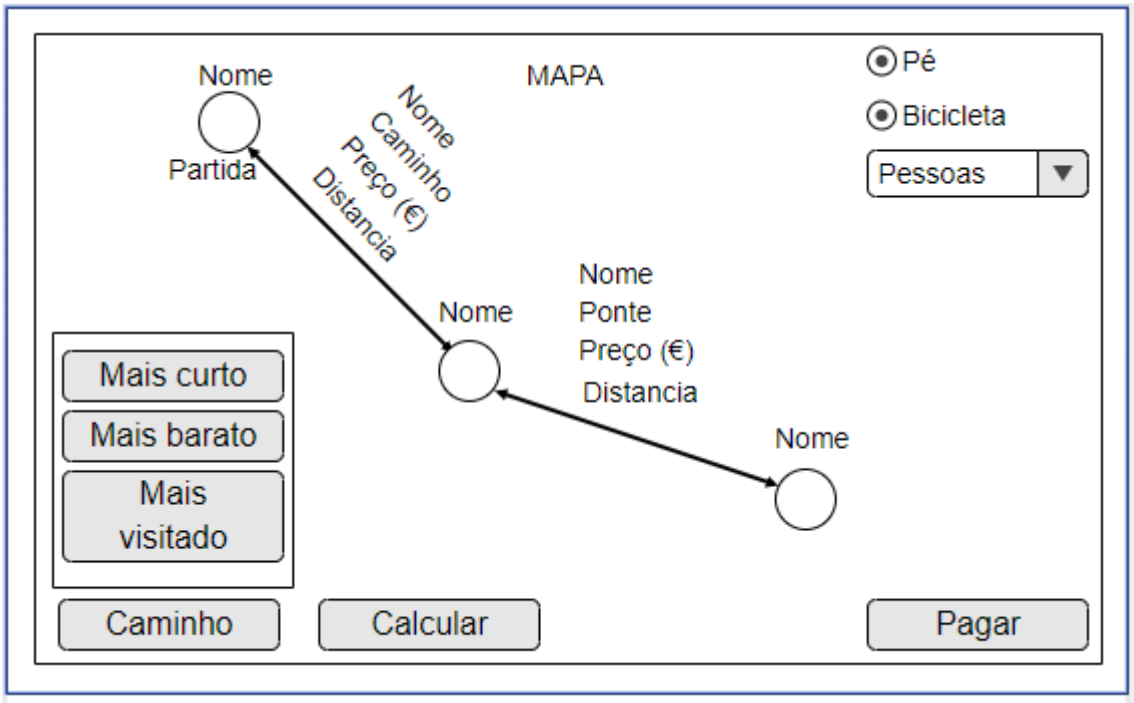


Figura 1 – *Mockup* do ecrã inicial

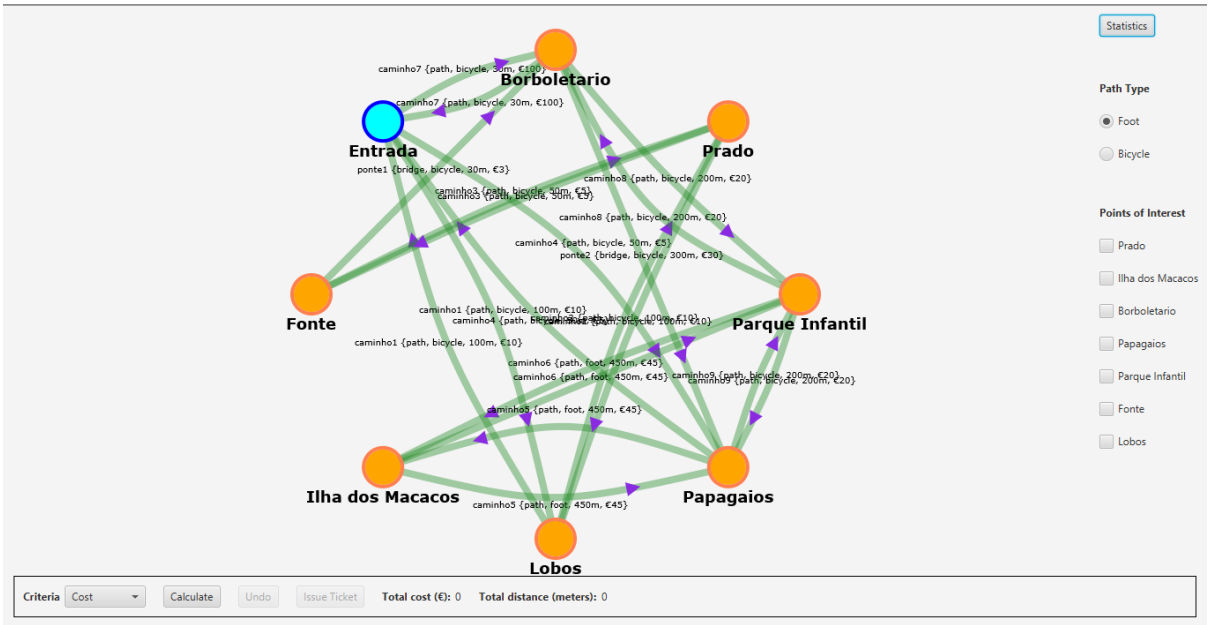


Figura 2 - Resultado final do ecrã inicial

Também foi contemplado nos *mockups* o ecrã / janela onde serão demonstradas e calculadas as estatísticas de acordo com as vendas para o parque / mapa em específico.

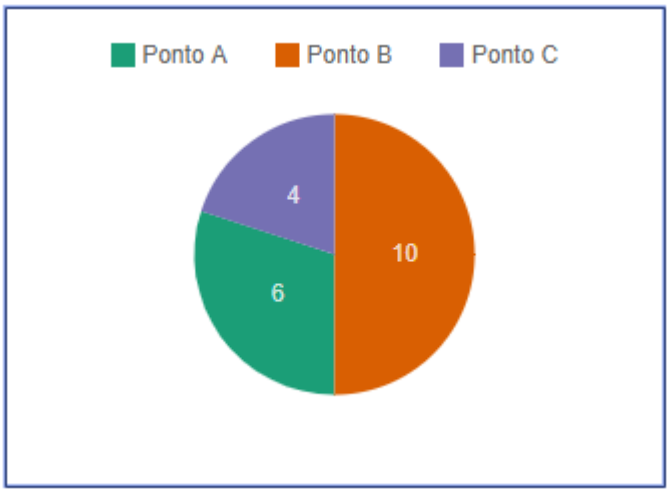


Figura 3 - *Mockup* da janela de estatísticas

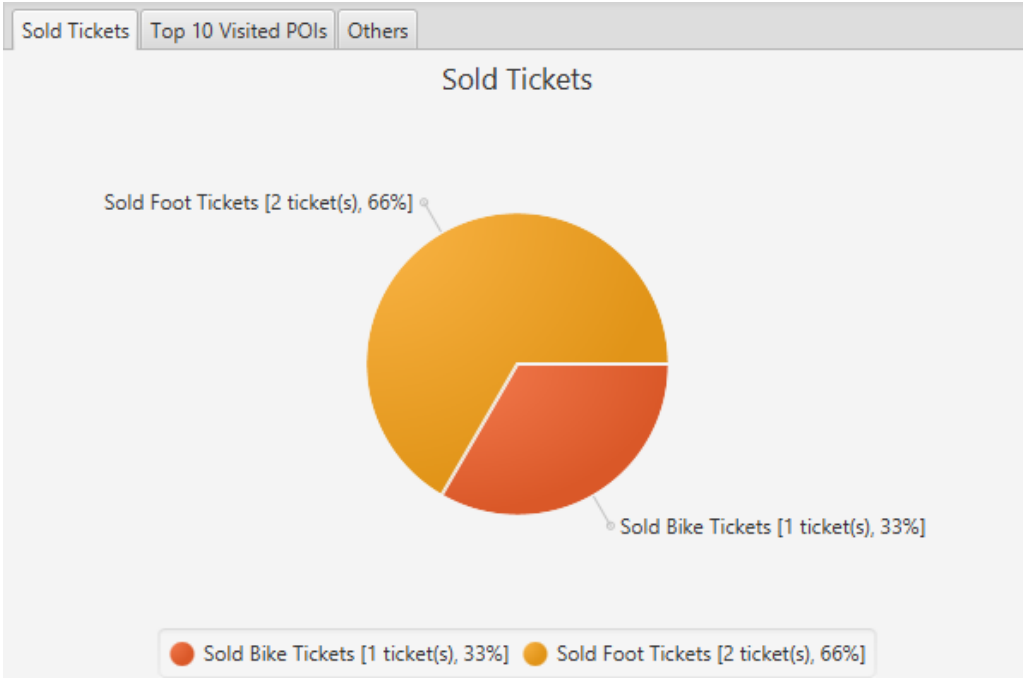


Figura 4 - Resultado final da janela de estatísticas

Por fim foi realizado o *mockup* da interface gráfica para a confirmação do pagamento, que irá detalhar com mais pormenor a informação referente ao trajeto posteriormente calculado.

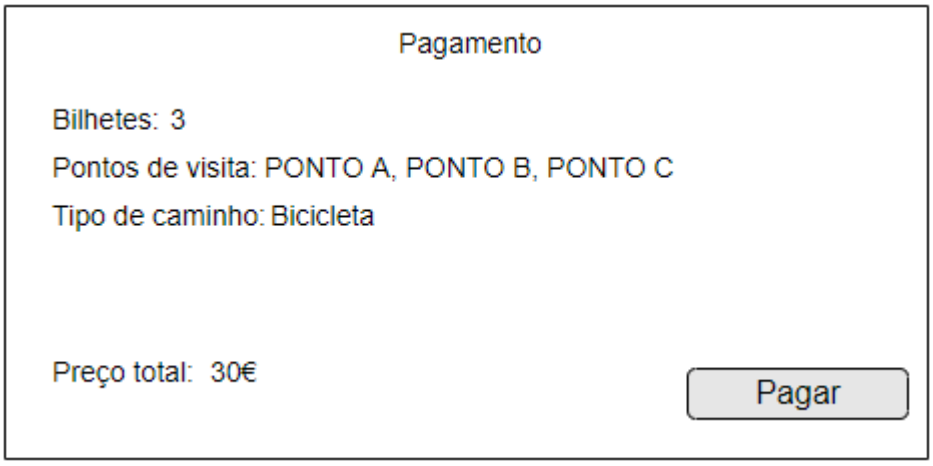


Figura 5 - *Mockup* da janela de confirmação de pagamento

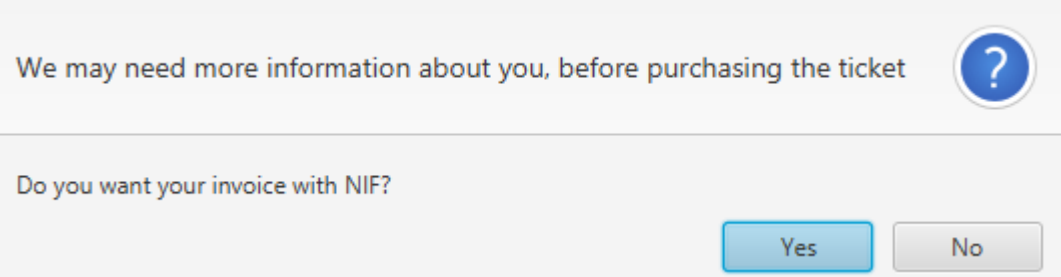


Figura 6 - Resultado final da janela de confirmação de pagamento

TIPOS ABSTRATOS DE DADOS IMPLEMENTADOS

Foi utilizado tipos abstratos de dados para o digraph implementado no projeto desenvolvido, para que fosse de fácil utilização na lógica de negócio do model da classe digraph.

Também foi utilizado tipos abstratos de dados na classe DocumentDAOSerialization para que posteriormente fosse utilizada tanto para a serialização dos documentos de bilhetes ou de recibos.

DIAGRAMA DE CLASSES

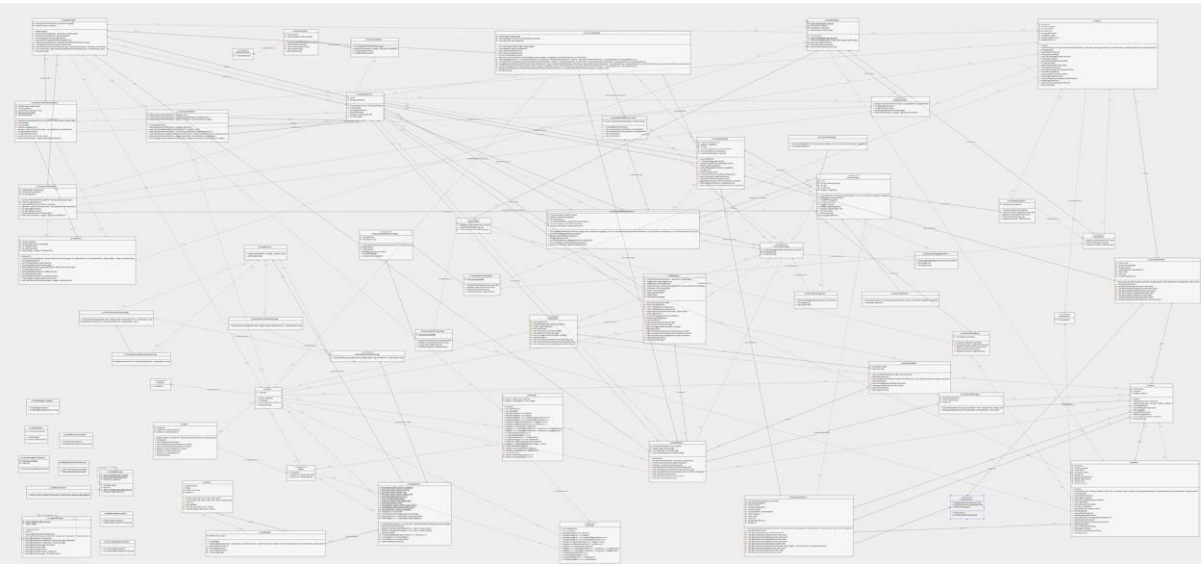


Figura 7 - Diagrama de classes

DESCRIÇÃO DAS CLASSES

Neste capítulo será explicado a função e responsabilidade de cada classe na aplicação desenvolvida.

CONFIGMANAGER

Esta é a classe responsável de recuperar as configurações diretamente do ficheiro de config, esta classe contém um método para inicializar todas as propriedades do ficheiro de configurações.

CUSTOMLOGGERFORMATTER

Esta classe tem como função formatar a string de saída para o ficheiro de log, desta forma faz com que o ficheiro seja mais facilmente visualizado pelo utilizador se assim o pretender.

DAOMANAGER

A classe DaoManager é responsável pela persistência de dados para os tickets emitidos, recibos e estatísticas dos mesmos. Esta classe é singleton por isso apenas é instanciada uma vez.

LOGGERMANAGER

Classe responsável pelo ficheiro de logs, onde serão escritos todos os erros ou funcionalidades despoletadas na aplicação. Esta função é importante não só para meios de debugging como também ser possível verificar o que realmente está a acontecer por de trás da interface gráfica.

MAINCONTROLLER

Esta classe é responsável pelos eventos despoletados na main view, fazendo chamadas ao backend / lógica das diversas funcionalidades.

MAINMODEL

Este é o modelo da main view, onde se encontra os métodos que irão buscar os dados necessários para mostrar ou calcular trajetos.

STATISTICSMODEL

O modelo da view das estatísticas que é responsável sobre a busca de dados para view de estatísticas. Nesta classe é onde se encontram os métodos para realizar estas mesmas buscas.

CALCULATEDDIJKSTRA

Esta classe é responsável por guardar os dados dos trajetos calculados com o algoritmo de dijkstra.

CALCULATEDPATH

Esta classe é responsável por guardar e manusear os dados dos trajetos calculados.

CALCULATEDPATHCARETAKER

Classe responsável por guardar os diversos estados da classe CalculatedPath, permitindo assim realizar undo.

CALCULATEDPATHMEMENTO

CONNECTION

É uma classe abstrata que é usada para diversos tipos de ligação.

CONNECTIONBRIDGE

Esta classe é responsável por guardar os dados do tipo de ponte.

CONNECTIONPATH

Esta classe é responsável por guardar os dados do tipo de caminho.

COURSEMANAGER

Esta classe irá realizar a manutenção e a gestão de todos trajetos calculados.

COURSEMANAGEREXCEPTION

Classe responsável por tratar e gerir exceções do cálculo de percursos.

CRITERIASTRATEGYCOST

É uma classe usada para calcular o custo de acordo com o critério de custo.

CRITERIASTRATEGYDISTANCE

É uma classe usada para calcular o custo de acordo com o critério de distância.

MAPMANAGER

Classe responsável pela criação do mapa e da criação do digraph.

MAPMANAGEREXCEPTION

Classe responsável por tratar e gerir exceções da geração do mapa.

POINTOFINTEREST

Classe responsável por guardar os dados dos pontos de interesse.

CLIENT

Classe responsável por guardar os dados dos clientes / utilizadores.

DOCUMENTMANAGER

É uma classe responsável pela criação dos documentos de bilhetes e recibos.

INVOICE

Classe para guardar os dados de recibos.

INVOICEDAOSERIALIZATION

Classe responsável pela persistência de dados com serialização (recibo).

INVOICEDAOSQLLITE

Classe responsável pela persistência de dados com SQLLite (recibo).

TICKET

Classe responsável por guardar os dados dos bilhetes.

TICKETDAOSERIALIZATION

Classe responsável pela persistência de dados com serialização (bilhete).

TICKETDAOSQLLITE

Classe responsável pela persistência de dados com SQLLite (bilhete).

STATISTICS

Classe responsável por guardar os dados das estatísticas.

STATISTICSDAOSERIALIZATION

Classe responsável pela persistência de dados com serialização (estatísticas).

STATISTICSDAOSQLLITE

Classe responsável pela persistência de dados com SQLLite (estatísticas).

CLIENTDIALOG

A classe referente à view do diálogo do formulário do cliente.

MAINVIEW

A classe referente à main view, aqui é onde todos os componentes gráficos da view ficam guardadas e inicializadas.

STATISTICSVIEW

Classe referente à view de estatísticas.

DIGRAPH

Classe de um digraph abstrato.

EDGE

Classe referente a uma aresta de um graph.

INVALIDEDGEEXCEPTION

Classe referente ao tratamento das exceções inválidas da aresta.

INVALIDVERTEXEXCEPTION

Classe referente ao tratamento das exceções inválidas do vertice.

VERTEX

Classe referente a uma aresta de um vertice.

PADRÕES DE SOFTWARE

No âmbito do conteúdo da matéria aprendida na unidade curricular de programação avançada, foi necessário seguir e usar padrões de software no desenvolvimento deste projeto, onde cada padrão de software utilizado, será explicado e justificado o uso do mesmo.

1. MODEL VIEW CONTROLER (MVC)

É uma boa prática de programação a divisão de responsabilidades para cada classe ou método. De modo a evitar misturar componentes gráficas com a lógica foi seguido o padrão de software MVC, este padrão passa pela divisão das funcionalidades de negócios, da seguinte forma:

- Model – Responsável pela persistência de dados, também dispara eventos para a *view*.
- View – Aqui é onde se encontram as componentes gráficas da interface da aplicação e que envia chamadas para o *controller*.
- Controller – Onde se encontra toda a lógica da *view.*

2. MEMENTO

Este padrão tem como objetivo guardar o estado de um objeto sem violar o encapsulamento do mesmo, para que seja possível retornar a esse estado em qualquer altura. Este foi o padrão utilizado para o "undo" dos trajetos calculados, permitindo desta forma que o utilizador consiga voltar a qualquer trajeto que tenho calculado anteriormente. Para seguir este padrão temos o objeto *Originator* que será o objeto com o estado pretendido restaurar, e uma classe *Caretaker* que é responsável por armazenar os estados dos objetos.

3. DATA ACCESS OBJECT (DAO)

Este padrão consiste na criação de um objeto de acesso de dados (DAO), cujo objetivo é abstrair e encapsular todos os acessos à fonte de dados, independente se esta é uma base de dados, um ficheiro JSON ou um ficheiro XML. Desta forma a responsabilidade do objeto DAO passa por gerir as ligações à fonte de dados. No projeto desenvolvido foi necessário a implementação de uma funcionalidade de presistencia de dados, no entanto, esta funcionalidade teria duas fontes de dados (JSON ou SQLite), sendo que seria necessário a utilização do padrão DAO para a realização desta funcionalidade.

4. STRATEGY

Strategy é categorizado como um padrão comportamental de desenvolvimento de software. De modo que delega as responsabilidades adquiridas pelas entidades, atribuindo, portanto, o comportamento. Assim a comunicação entre os objetos é aprimorada, pois há a distribuição das responsabilidades. Este padrão serve para certos casos, como quando existem várias classes que diferem apenas no seu comportamento ou diferentes variantes de um algoritmo. Este foi utilizado no cálculo de diferentes trajetos (a pé ou de bicicleta, por exemplo).

5. SINGLETON

O padrão Singleton garante que um dado objeto ou classe só tenha apenas uma única instância e que seja criado um ponto de acesso global a essa instância. Este padrão garante a existência de apenas uma instância de uma classe, mantendo um ponto global de acesso ao seu objeto. Sendo que este foi utilizado na classe responsável pela persistência dos dados de fatura, bilhetes e estatísticas, fazendo com que a ligação à fonte de dados fosse apenas realizada uma vez. Também foi utilizado para a classe responsável pelos *logs*.

6. OBSERVER

O padrão Observer permite que objetos interessados sejam avisados da mudança de estado ou de outros eventos ocorridos num outro objeto. Desta forma permite que as classes das componentes gráficas sejam observadas. Este padrão foi utilizado em conjunto com o padrão MVC.

7. ITERATOR

Este padrão serve para percorrer os elementos de uma coleção. O Iterator em Java é uma interface, que define os métodos next() e hasNext(). Sendo que este padrão é utilizado na classe do *digraph* do projeto, dando assim a vantagem de percorrer os elementos do mapa.

8. TEMPLATE METHOD

Este método serve para quando existem algoritmos com uma estrutura semelhante em várias subclasses da mesma estrutura de classes. Tendo duplicação de Código. Permite evitar a duplicação de código. No entanto são difíceis de gerir quando o template método tem muitos passos. Este padrão foi utilizado no âmbito do método toString() na classe da coneção.

REFACTORING

Bad smells

Bad Smells	Número de situações	Refactoring
Duplicated code	7	Criar um template método (Figura 8, 9, 10) e PullupMethod
Long Method	3	Extract Method (Figura 11)

Long Class	2	ExtractSubclass

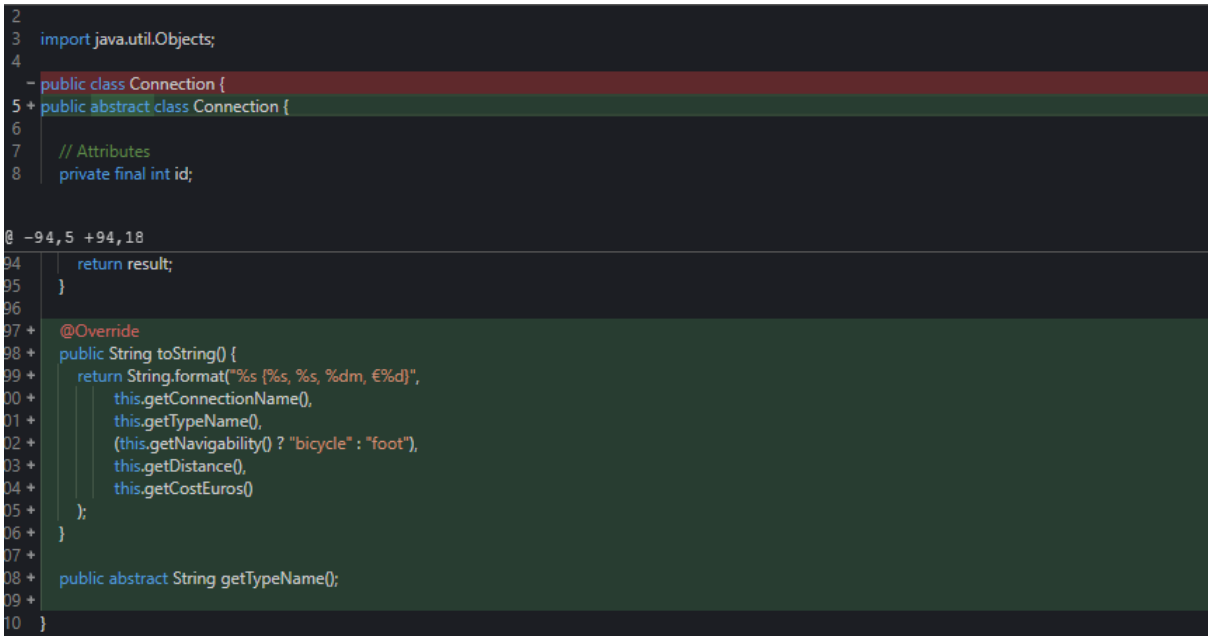
```
2
3   import java.util.Objects;
4
-   public class Connection {
5   + public abstract class Connection {
6
7       // Attributes
8       private final int id;


@\@ -94,5 +94,18
94          return result;
95      }
96
97  +   @Override
98  +   public String toString() {
99  +       return String.format("%s {%s, %s, %dm, €%d}",
100 +               this.getConnectionName(),
101 +               this.getTypeName(),
102 +               (this.getNavigability() ? "bicycle" : "foot"),
103 +               this.getDistance(),
104 +               this.getCostEuros()
105 +       );
106 +   }
107 +
108 +   public abstract String getTypeName();
109 +
110 }
```

Figura 8 - Template method exemplo (1)

```
14      }
15
16      @Override
-       public String toString() {
-           return String.format("%s {bridge, %dm, €%d%s}",
-                   getConnectionName(), getDistance(), getCostEuros(), (getNavigability() ? ", bicycle" : ""));
17  +   public String getTypeName() {
18  +       return "bridge";
19      }
20
21  }
```

Figura 9 - Template method exemplo (2)



```
2
3   import java.util.Objects;
4
-   public class Connection {
5 +  public abstract class Connection {
6
7       // Attributes
8       private final int id;


@ -94,5 +94,18
94          return result;
95      }
96
97 +    @Override
98 +    public String toString() {
99 +        return String.format("%s {%s, %s, %dm, €%d}",
00 +                this.getConnectionName(),
01 +                this.getTypeName(),
02 +                (this.getNavigability() ? "bicycle" : "foot"),
03 +                this.getDistance(),
04 +                this.getCostEuros()
05 +        );
06 +    }
07 +
08 +    public abstract String getTypeName();
09 +
10  }
```

Figura 10 - Template method (3)

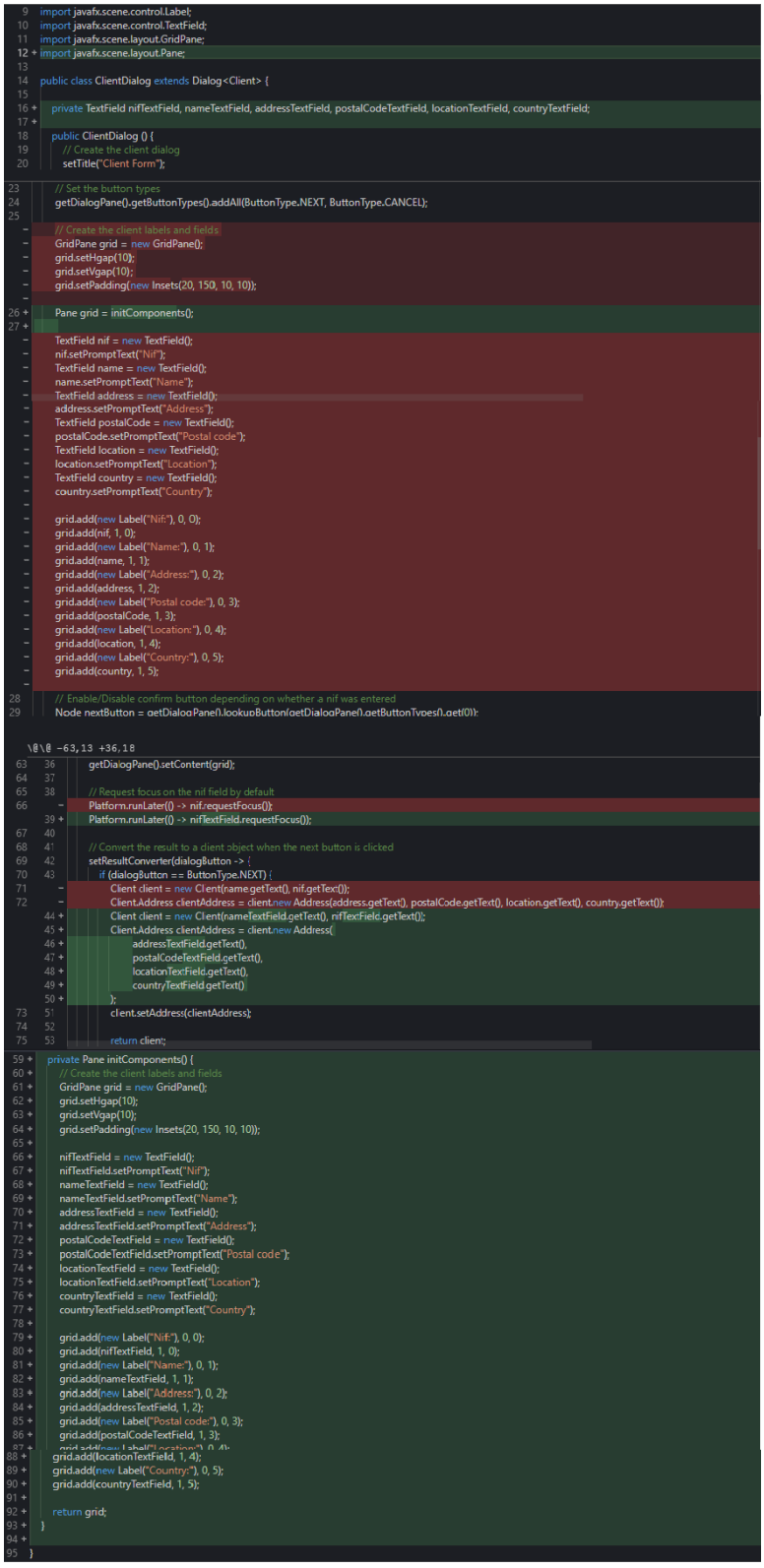


Figura 11 - Extract method