

Programação Avançada 2018/2019

Licenciatura em Eng^a. 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.

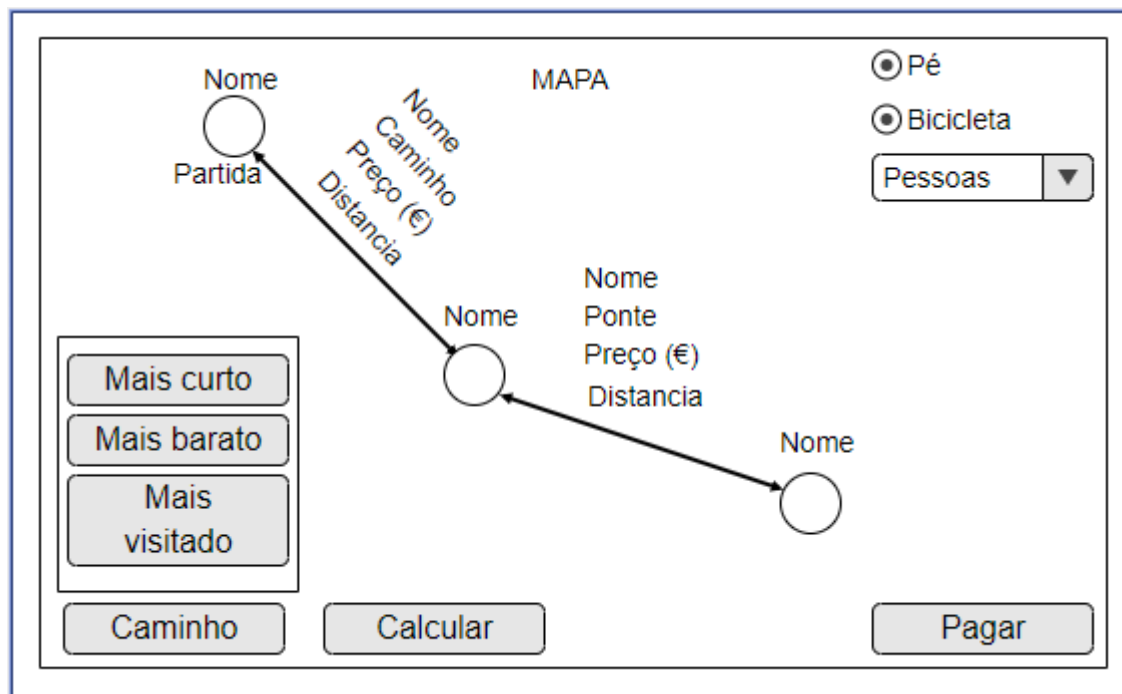


Figura 1 – Mockup do ecrã inicial

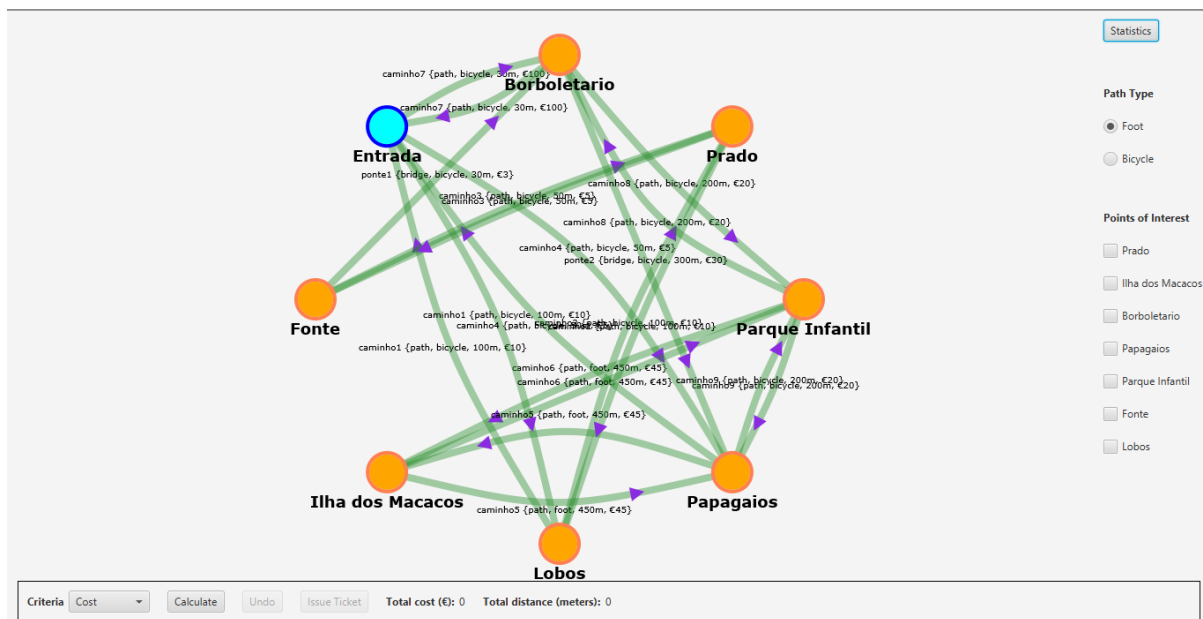


Figura 2 - Resultado final do ecrã inicial

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.

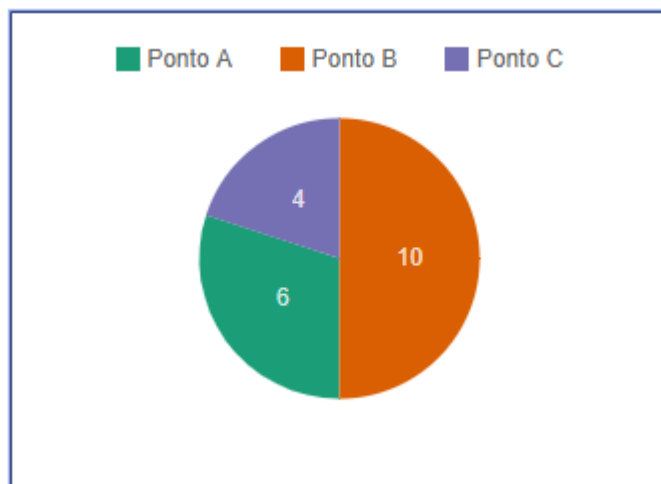


Figura 3 - Mockup da janela de estatísticas

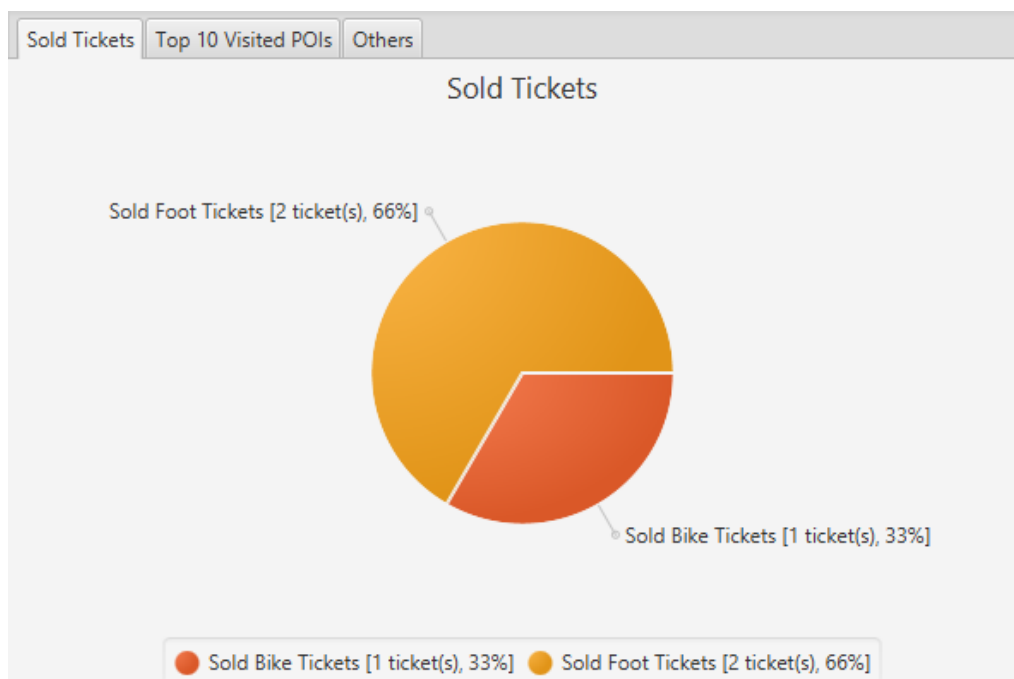


Figura 4 - Resultado final da janela de estatísticas

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.

The mockup shows a window titled "Pagamento" (Payment). It contains the following text: "Bilhetes: 3", "Pontos de visita: PONTO A, PONTO B, PONTO C", and "Tipo de caminho: Bicicleta". At the bottom left, it states "Preço total: 30€". On the bottom right, there is a button labeled "Pagar".

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

We may need more information about you, before purchasing the ticket

Do you want your invoice with NIF?

Yes No

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

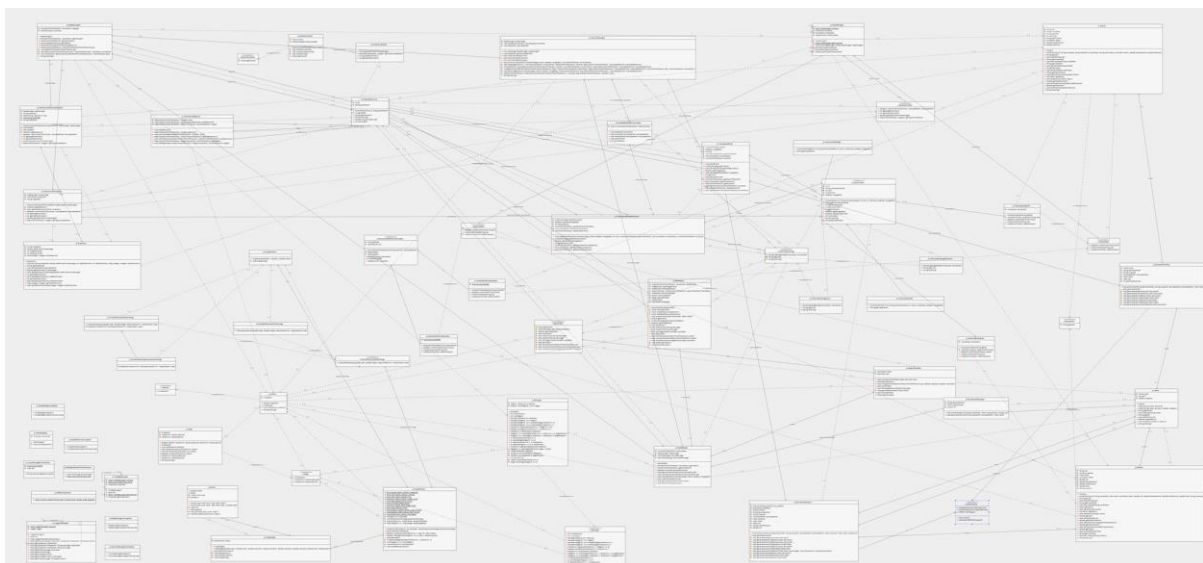


Figura 7 - Diagrama de classes

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 SQLite (recibo).

TICKET

Classe responsável por guardar os dados dos bilhetes.

TICKETDAOSerialization

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

TICKETDAOSQLite

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

STATISTICS

Classe responsável por guardar os dados das estatísticas.

STATISTICSDAOSerialization

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

STATISTICSDAOSQLite

Classe responsável pela persistência de dados com SQLite (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 tenha 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 persistência 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 conexão.

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
5 - public class Connection {
6 + public abstract class Connection {
7
8 // Attributes
9 private final int id;
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94 return result;
95 }
96
97 + @Override
98 + public String toString() {
99 + return String.format("%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
17 + public String toString() {
18 + return String.format("%s {bridge, %dm, €%d%s}",
19 + getConnectionName(), getDistance(), getCostEuros(), (getNavigability() ? ", bicycle" : ""));
20 + }
21 + public String getTypeName() {
22 + return "bridge";
23 + }
24 }
```

Figura 9 - Template method exemplo (2)

```

2
3 import java.util.Objects;
4
5 - public class Connection {
6 + public abstract class Connection {
7
8     // Attributes
9     private final int id;
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Figura 10 - Template method (3)

```

9 import javafx.scene.control.Label;
10 import javafx.scene.control.TextField;
11 import javafx.scene.layout.GridPane;
12 import javafx.scene.layout.Pane;
13
14 public class ClientDialog extends Dialog<Client> {
15
16     private TextField nifTextField, nameTextField, addressTextField, postalCodeTextField, locationTextField, countryTextField;
17
18     public ClientDialog() {
19         // Create the client dialog
20         setTitle("Client Form");
21
22         // Set the button types
23         getDialogPane().getButtonTypes().addAll(ButtonType.NEXT, ButtonType.CANCEL);
24
25         // Create the client labels and fields
26         GridPane grid = new GridPane();
27         grid.setHgap(10);
28         grid.setVgap(10);
29         grid.setPadding(new Insets(20, 150, 10, 10));
30
31         Pane grid = initComponents();
32
33         TextField nif = new TextField();
34         nif.setPromptText("Nif");
35         TextField name = new TextField();
36         name.setPromptText("Name");
37         TextField address = new TextField();
38         address.setPromptText("Address");
39         TextField postalCode = new TextField();
40         postalCode.setPromptText("Postal code");
41         TextField location = new TextField();
42         location.setPromptText("Location");
43         TextField country = new TextField();
44         country.setPromptText("Country");
45
46         grid.add(new Label("Nif:"), 0, 0);
47         grid.add(nif, 1, 0);
48         grid.add(new Label("Name:"), 0, 1);
49         grid.add(name, 1, 1);
50         grid.add(new Label("Address:"), 0, 2);
51         grid.add(address, 1, 2);
52         grid.add(new Label("Postal code:"), 0, 3);
53         grid.add(postalCode, 1, 3);
54         grid.add(new Label("Location:"), 0, 4);
55         grid.add(location, 1, 4);
56         grid.add(new Label("Country:"), 0, 5);
57         grid.add(country, 1, 5);
58
59         // Enable/Disable confirm button depending on whether a nif was entered
60         Node nextButton = getDialogPane().lookupButton(getDialogPane().getButtonTypes().get(0));
61
62         \0\0 -63,13 +36,18
63         getDialogPane().setContent(grid);
64
65         // Request focus on the nif field by default
66         Platform.runLater(() -> nif.requestFocus());
67         Platform.runLater(() -> nifTextField.requestFocus());
68
69         // Convert the result to a client object when the next button is clicked
70         setResultConverter(dialogButton -> {
71             if (dialogButton == ButtonType.NEXT) {
72                 Client client = new Client(name.getText(), nif.getText());
73                 ClientAddress clientAddress = client.new Address(address.getText(), postalCode.getText(), location.getText(), country.getText());
74                 Client client = new Client(nameTextField.getText(), nifTextField.getText());
75                 ClientAddress clientAddress = client.new Address(
76                     addressTextField.getText(),
77                     postalCodeTextField.getText(),
78                     locationTextField.getText(),
79                     countryTextField.getText()
80                 );
81                 client.setAddress(clientAddress);
82                 return client;
83             }
84         });
85
86     private Pane initComponents() {
87         // Create the client labels and fields
88         GridPane grid = new GridPane();
89         grid.setHgap(10);
90         grid.setVgap(10);
91         grid.setPadding(new Insets(20, 150, 10, 10));
92
93         nifTextField = new TextField();
94         nifTextField.setPromptText("Nif");
95         nameTextField = new TextField();
96         nameTextField.setPromptText("Name");
97         addressTextField = new TextField();
98         addressTextField.setPromptText("Address");
99         postalCodeTextField = new TextField();
100        postalCodeTextField.setPromptText("Postal code");
101        locationTextField = new TextField();
102        locationTextField.setPromptText("Location");
103        countryTextField = new TextField();
104        countryTextField.setPromptText("Country");
105
106        grid.add(new Label("Nif:"), 0, 0);
107        grid.add(nifTextField, 1, 0);
108        grid.add(new Label("Name:"), 0, 1);
109        grid.add(nameTextField, 1, 1);
110        grid.add(new Label("Address:"), 0, 2);
111        grid.add(addressTextField, 1, 2);
112        grid.add(new Label("Postal code:"), 0, 3);
113        grid.add(postalCodeTextField, 1, 3);
114        grid.add(new Label("Location:"), 0, 4);
115        grid.add(locationTextField, 1, 4);
116        grid.add(new Label("Country:"), 0, 5);
117        grid.add(countryTextField, 1, 5);
118
119        return grid;
120    }
121 }

```

Figura 11 - Extract method