Universitat de Barcelona

Facultat de Matemàtiques i Enginyeria Informàtica

Pràctica 4

Grau en Enginyeria Informàtica

Curs 2022-2023 | *Cinquè Semestre*

Disseny de Software (DS)

Autor:
Mario Vilar (20392724)
David Díez (20392643)

Professor:
Leandro Zardaín

24 de desembre de 2022

Presentació de la pràctica

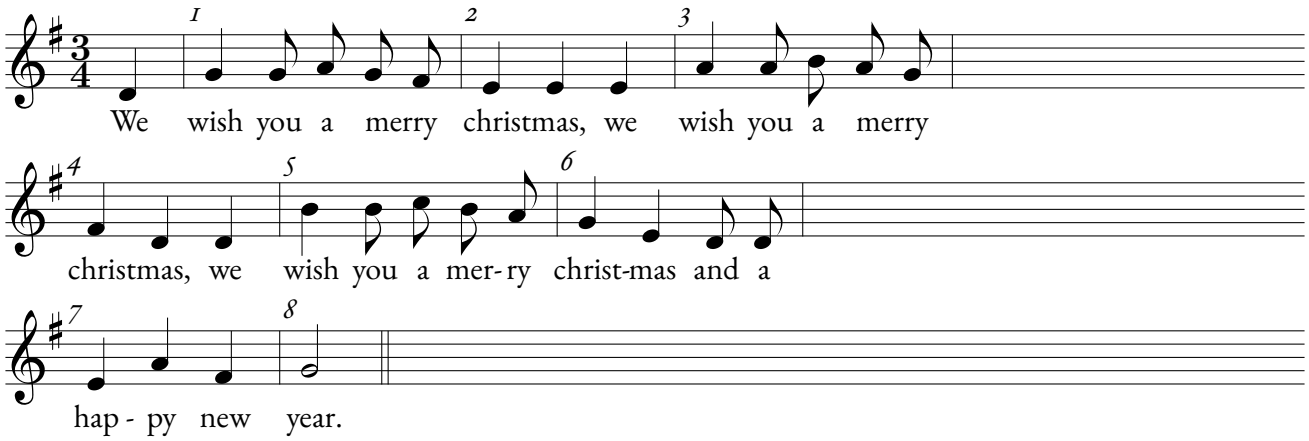
TripUB: App per mòbil per planificar fàcilment viatges i rutes.

Universitat de Barcelona

Índex

I

Introducció



En aquesta pràctica se'ns demanava desenvolupar una sèrie de testos per a un club d'activitats *TripUB*, una app per ordinador de sobretaula per poder planificar fàcilment viatges i rutes.

L'objectiu d'aquesta pràctica és, en essència, començar a entendre el paradigma de Programació Orientada a Testos per tal de poder modelar, mitjançant la metodologia AGILE, la interfície de l'usuari amb el conjunt de classes i informació que se'ns dona. En efecte, caldrà gestionar rutes, allotjaments i clients.

1. Inicialment l'aplicació TripUB disposa ja de rutes ja definides i la usuària prèviament enregistrada a l'aplicació pot accedir-hi, explorar-les i modificar-les lleugerament al seu gust, així com reservar els allotjaments que ofereix l'aplicació.
2. Només obrir l'app de TripUB, s'ha de crear un nou compte com a nova persona enregistrada a l'aplicació o bé s'entra directament al compte, sense necessitat de fer login, ja que el sistema ja té guardades les seves credencials en el mòbil.
3. Quan la usuària crea un compte a l'app TripUB s'utilitza l'adreça de correu electrònic per a identificar-la de forma inequívoca, i a més, es demana el seu DNI o NIE, nom, cognoms, any de naixement i la seva adreça completa, la qual inclou arrer, nombre, codi postal i població.
4. L'aplicació permet planificar tant viatges de rutes llargues que consten de diferents etapes on es pernocta, com rutes més curtes, o tracks, per a excursions o passejades. També es poden fer combinacions de rutes llargues amb excursions entremig. L'aplicació recomana un cert transport per a realitzar un tram

(Cotxe, Bicicleta o A Peu), tot i que després, durant la planificació, es pot canviar de mitjà de transport que es voldrà fer servir. Per qualsevol mitjà de transport es vol tenir la velocitat mitja per fer una estimació del temps que es trigarà en fer algun recorregut.

5. Així, l'app TripUB proporciona rutes que comencen i acaben en certes localitats situades per les comarques de Catalunya. Les rutes poden estar formades per diferents tipus de trams (trams entre etapes o trams de track). Quan es tracta d'una etapa, l'aplicació també facilita allotjaments de diferents tipus (hotels, cases rurals, càmpings o refugis), que es diferencien pel seu grau de comoditat, el preu però també segons si ofereixen o no esmorzar, Mitja Pensió o Pensió Completa. Els hotels ofereixen els tres tipus d'àpats, les cases rurals no ofereixen dinar, els refugis només ofereixen esmorzars i en el càmping no es dona cap tipus de servei. Aquesta informació es té en compte per a calcular el cost total de la ruta així com el preu per persona i nit. Pel cost total de la ruta també es té en compte el cost del transport, si és el cas. Per exemple, si s'utilitza el cotxe es té la informació dels litres que gasta el cotxe als 100 kms i el preu de la gasolina.
6. Cada ruta té un nom, un identificador únic i una descripció. A la ruta es guarda també el seu nivell de dificultat (Alt, Mitjà o Baix) i si és circular o no, com a informació addicional a la distància total, el nombre de dies i el seu cost estimat, per permetre filtres en les consultes. La distància total es calcula segons la distància entre trams.
7. Cada ruta té un nom, un identificador únic i una descripció. A la ruta es guarda també el seu nivell de dificultat (Alt, Mitjà o Baix) i si és circular o no, com a informació addicional a la distància total, el nombre de dies i el seu cost estimat, per permetre filtres en les consultes. La distància total es calcula segons la distància entre trams.
8. En el cas que la ruta impliqui etapes en les què es fa nit, quan es planifica, cal reservar un dels allotjaments proposats per l'app. En l'aplicació, també es permet fer grups de persones que pensen realitzar juntes la ruta.

L'app TripUB a dissenyar assumeix que la introducció de les dades de les rutes ja s'ha realitzat. Així, l'app es centrarà a gestionar l'accés a la informació disponible (diferents consultes que permetin triar una ruta), així com fer grups i sortir d'un grup, planificar la ruta, reservar allotjaments, si és necessari, i calcular els costos. A part en el perfil, es podrà consultar l'històric de rutes fetes o les guardades per més endavant.

En relació a les consultes, inicialment l'app només oferirà consultes per obtenir les rutes per comarca o per població. En les cerques es mostra un resum de la ruta i la usuària pot clicar per veure més informació, i en el cas que estigui interessada, la pot seleccionar per planificar la seva pròpia experiència.

Ens hem assegurat de comentar bé tot el nostre codi, així que no ens entretindrem amb això.

Llista de codis font o listings

II

Cos del treball

2.1

Solució de codi

Com que hi ha moltes classes, i algunes d'aquestes són molt evidents donat el model de domini, aquí posarem les més importants; en efecte, hi ha una correspondència molt alta entre la quantitat de codi que hem editat en una classe i la importància d'aquesta. Evidentment es podran trobar totes als fitxers que es lliuraran de la pràctica.

```java
package ub.edu.controller;

import javafx.scene.control.Alert;
import ub.edu.model.*;
import ub.edu.resources.ResourcesFacade;

import java.util.*;


public class Controller {

    private static Controller controller;

    private ResourcesFacade inicialitzador;

    // private ModelFacade modelFacade;

    private TramFacade tramFacade;
    private AllotjamentFacade allotjamentFacade;
    private PersonFacade personFacade;
    private LocalitatsFacade localitatsFacade;
    private RutaFacade rutaFacade;

    private TripUB tripUB;

    private SessionMemory sessionMemory;

    private Controller() {
        tripUB = TripUB.getInstance();
        sessionMemory = SessionMemory.getInstance();

```

```
32          // modelFacade = ModelFacade.getInstance(tripUB);
33          tramFacade = TramFacade.getInstance(tripUB);
34          allotjamentFacade = AllotjamentFacade.getInstance(tripUB);
35          personFacade = PersonFacade.getInstance(tripUB);
36          localitatsFacade = LocalitatsFacade.getInstance(tripUB);
37          rutaFacade = RutaFacade.getInstance(tripUB);
38
39          // inicialitzador = ResourcesFacade.getInstance(tripUB, modelFacade);
40          inicialitzador = ResourcesFacade.getInstance(tripUB, tramFacade,allotjamentFacade,personFacade,localitatsFacade,rutaFacade);
41          inicialitzador.populateTripUB();
42
43      }
44
45      public static Controller getInstance() {
46          if(controller == null) {
47              controller = new Controller();
48          }
49          return controller;
50      }
51
52      public SessionMemory getSessionMemory() {
53          return sessionMemory;
54      }
55
56      public List<HashMap<Object,Object>> getAllRutesPerNom() {
57          return rutaFacade.getAllRutesPerNom();
58      }
59
60      public List<HashMap<Object,Object>> getAllGrupsPerNom() {
61          return personFacade.getAllGrupsPerNom();
62      }
63
64      public String validateRegistrePersona(String username, String password) {
```

```
        switch (personFacade.validateRegistrePersona(username, password)) {
            case REGISTRE_VALID:
                return "Registre vàlid";
            case PERSONA_DUPLICADA:
                return "Persona duplicada";
            case FORMAT_INCORRECTE_CORREU_PWD:
                return "Format incorrecte";
            default:
        }
        return"Cas no contemplat";
    }

    public String loguejarPersona(String username, String password){

        switch (personFacade.loguejarPersona(username, password)) {
            case 0:
                return StatusType.REGISTRE_VALID.toString();
            case 1:
                return StatusType.CORREU_INEXISTENT.toString();
            case 2:
                 return StatusType.CONTRASENYA_INCORRECTA.toString();
             default:
        }
        return"Cas no contemplat";
    }

    public String recuperarContrasenya(String username){
        return personFacade.recuperarContrasenya(username);
    }

    public StatusType addNewPersona(String correu, String nom, String cognoms, String dni, String password, Integer id_tripUB) throws Exception
        return inicialitzador.addNewPersona(correu, nom, cognoms, dni, password, id_tripUB);
    }
```

```
98
99      public StatusType loguejarSociStatus(String correu, String password){
100         return (personFacade.loguejarSociStatus(correu, password));
101     }
102
103     public Integer addNewGrup(String grupNom) throws Exception {
104         return inicialitzador.addNewGrup(grupNom);
105     }
106
107     public boolean addRelacionGrupPersona(String correuPersona, Integer idGrup) throws Exception {
108        return  inicialitzador.addRelacionGrupPersona(correuPersona, idGrup);
109
110     }
111
112     public ArrayList<HashMap<Object,Object>> getAllGrupsPerPersona(String correu) throws Exception {
113         return personFacade.getAllGrupsPerPersona(correu);
114     }
115
116     public List<HashMap<Object, Object>> getAllComarques() throws Exception {
117         return localitatsFacade.getAllComarques();
118     }
119
120     public List<HashMap<Object,Object>> getAllLocalitats() throws Exception {
121         return localitatsFacade.getAllLocalitats();
122     }
123
124     public List<HashMap<Object,Object>> getAllTramsEtapaTrackByRutaId(Integer id_ruta) throws Exception {
125         return tramFacade.getAllTramsEtapaTrackByRutaId(id_ruta);
126
127     }
128
129     public List<HashMap<Object, Object>> llistarRutesPerLocalitat(String nomLocalitat) {
130         return rutaFacade.llistarRutesPerLocalitat(nomLocalitat);
```

```
    }

    public List<HashMap<Object, Object>> llistarRutesPerComarques(String nomComarca) {
        return rutaFacade.llistarRutesPerComarca(nomComarca);
    }

    public HashMap<Object,Object> getTramTrackById(Integer id) throws Exception {
        return tramFacade.getTramTrackById(id);
    }

    public HashMap<Object,Object> getLocalitat_y_PuntDePasById(Integer id) throws Exception {
        return localitatsFacade.getLocalitat_y_PuntDePasById(id);
    }


    public HashMap<Object, Object> getTramEtapaById(Integer id) throws Exception {
        return tramFacade.getTramEtapaById(id);
    }

    public HashMap<Object, Object> getLocalitat_y_EtapaByID(Integer id) throws Exception {
        return localitatsFacade.getLocalitat_y_EtapaByID(id);
    }

    public List<HashMap<Object,Object>> getAllAllotjaments() throws Exception {
        return allotjamentFacade.getAllAllotjaments();
    }

    public HashMap<Object, Object> getRutaById(Integer id_ruta) throws Exception {
        return rutaFacade.getRutaById(id_ruta);
    }

    public HashMap<Object,Object> getLocalitatByID(Integer id) throws Exception {
        return localitatsFacade.getLocalitatByID(id);
```

```
164     }
165
166     public Integer getIdGrupByName(String newValue) {
167         return tripUB.getIdGrupByName(newValue);
168     }
169
170
171     public String validatePassword(String b) {
172         switch (Seguretat.validatePassword(b)) {
173             case CONTRASENYA_NO_SEGURA:
174                 return "Contrasenya no prou segura";
175             case CONTRASENYA_SEGURA:
176                 return "Contrasenya segura";
177         }
178         return "Cas no contemplat";
179     }
180
181     public String validateUsername(String a) {
182         switch (Seguretat.validateUsername(a)) {
183             case CORREU_INCORRECTE:
184                 return "Correu en format incorrecte";
185             case CORREU_CORRECTE:
186                 return "Correu en format correcte";
187         }
188         return "Cas no contemplat";
189
190     }
191
192     private String exceptionTreatment(Exception e) {
193         Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
194         alert.setAlertType(Alert.AlertType.ERROR);
195         alert.setTitle("Error");
196         alert.setHeaderText("ERROR DEL SISTEMA!");
```

```
197         alert.setContentText(e.getMessage());
198         alert.showAndWait();
199         return e.getMessage();
200     }
201 
202     // el controlador o la façana ha de notificar a la vista que ha canviat, per poder actualitzar la taula
203     public String reservarAllotjament(Integer idRuta, Integer idTram, Integer idAllotjament, String correuPersona,
204                                       String tipusPagament, int numNits) {
205         try {
206             Reserva reserva = allotjamentFacade.crearReserva(idRuta, idTram, idAllotjament, correuPersona, tipusPagament, numNits);
207             inicialitzador.addReserva(reserva); /* ens crearà el id de la reserva */
208             return allotjamentFacade.reservarAllotjament(reserva);
209         }
210         catch (Exception e) {
211             return exceptionTreatment(e);
212         }
213     }
214 
215     public String anularReserva(Integer nomRuta, String nomPersona) {
216         try {
217             Reserva reserva = allotjamentFacade.anularReserva(nomRuta, nomPersona);
218             inicialitzador.removeReserva(reserva);
219             return "Hem processat la cancelació de la teva reserva correctament";
220         }
221         catch (Exception e) {
222             return exceptionTreatment(e);
223         }
224     }
225 
226     public Iterable<HashMap<Object,Object>> getAllotjamentsPagaments() {
227         return allotjamentFacade.getAllotjamentsPagaments();
228     }
229 
```

```
230     public Iterable<HashMap<Object,Object>> getAllotjamentByPagament(String tipusPagament) {
231         return allotjamentFacade.getAllotjamentByPagament(tipusPagament);
232     }
233 
234 }
```

Listing 1: Classe *Controlador* del nostre projecte.

```
1  package ub.edu.model;
2
3  import ub.edu.model.pagament.Pagament;
4
5  import java.time.LocalDate;
6
7  public class Reserva {
8      private Integer id;
9      private LocalDate dateCheckIn;
10     private LocalDate dateCheckOut;
11
12     private String tipusPagament; // "Anticipat"....
13     private String correuPersona;
14     private Integer idAllotjament;
15
16     private Integer idRuta;
17
18     private Integer idTram;
19
20     public Reserva() {
21     }
22
23     public Reserva(LocalDate dateCheckIn, LocalDate dateCheckOut,
24                    String tipusPagament, String correu_persona, Integer id_allotjament) {
```

```
        this();
        this.dateCheckIn = dateCheckIn;
        this.dateCheckOut = dateCheckOut;
        this.tipusPagament = tipusPagament;
        this.correuPersona = correu_persona;
        this.idAllotjament = id_allotjament;
    }

    public Reserva(String correuPersona, Integer idRuta, Integer idTram, LocalDate dataCreacio, LocalDate dataFinal,
                   Integer idAllotjament, String tipusPagament) {
        setCorreuPersona(correuPersona);
        setIdRuta(idRuta);
        setIdTram(idTram);
        setDateCheckIn(dataCreacio);
        setDateCheckOut(dataFinal);
        setIdAllotjament(idAllotjament);
        setTipusPagament(tipusPagament);
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public LocalDate getDateCheckIn() {
        return dateCheckIn;
    }

    public void setDateCheckIn(LocalDate dateCheckIn) {
        this.dateCheckIn = dateCheckIn;
```

```
58      }
59
60      public LocalDate getDateCheckOut() {
61          return dateCheckOut;
62      }
63
64      public void setDateCheckOut(LocalDate dateCheckOut) {
65          this.dateCheckOut = dateCheckOut;
66      }
67
68      public String getTipusPagament() {
69          return tipusPagament;
70      }
71
72      public void setTipusPagament(String tipusPagament) {
73          this.tipusPagament = tipusPagament;
74      }
75
76      public String getCorreuPersona() {
77          return correuPersona;
78      }
79
80      public void setCorreuPersona(String correuPersona) {
81          this.correuPersona = correuPersona;
82      }
83
84      public Integer getIdAllotjament() {
85          return idAllotjament;
86      }
87
88      public void setIdAllotjament(Integer idAllotjament) {
89          this.idAllotjament = idAllotjament;
90      }
```

```
91
92      @Override
93      public String toString() {
94          return "Reserva{" +
95                  "id=" + id +
96                  ", dateCheckIn=" + dateCheckIn +
97                  ", dateCheckOut=" + dateCheckOut +
98                  ", tipusPagament='" + tipusPagament + '\'' +
99                  ", correu_persona='" + correuPersona + '\'' +
100                 ", id_allotjament=" + idAllotjament +
101                 '}';
102     }
103
104     public void setIdRuta(Integer idRuta) {
105         this.idRuta = idRuta;
106     }
107
108     public void setIdTram(Integer idTram) {
109         this.idTram = idTram;
110     }
111
112     private Pagament getPagament() throws Exception{
113         String name = Pagament.class.getPackage().getName();
114
115         Pagament pagament = (Pagament)Class.forName(name + "."+ this.tipusPagament).getDeclaredConstructor().newInstance();
116
117         pagament.setDataInici(getDateCheckIn());
118
119         return pagament;
120     }
121
122
123     public double getPreu(double preuTotal) throws Exception {
```

```
124         return getPagament().ferPagament(preuTotal);
125     }
126
127     // mirem si podem anul·lar la reserva
128     public boolean getAnulacio() throws Exception {
129         return getPagament().ferAnulacio();
130     }
131
132     public String getInfoPagament() throws Exception {
133         return getPagament().toString();
134     }
135
136     public String getInfo(double preuTotal) throws Exception {
137         return getPagament().display(preuTotal);
138     }
139
140 }
```

Listing 2: Classe *Reserva* del nostre projecte.

```
1  package ub.edu.view;
2
3  import javafx.beans.property.SimpleStringProperty;
4  import javafx.beans.value.ChangeListener;
5  import javafx.beans.value.ObservableValue;
6  import javafx.fxml.FXML;
7  import javafx.geometry.Pos;
8  import javafx.scene.Node;
9  import javafx.scene.control.*;
10 import javafx.scene.control.cell.PropertyValueFactory;
11 import javafx.scene.image.Image;
12 import javafx.scene.image.ImageView;
```

```
import javafx.scene.input.MouseButton;
import javafx.scene.layout.AnchorPane;
import javafx.scene.text.Text;
import ub.edu.controller.Controller;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

public class EscenaMain extends Escena implements SceneObserver {
    private static final double ESPAI_ENTRE_BOTONS = 30;

    private Controller controller;

    public Button ruta_btn;
    public Button button_izq_resetFiltres_main;
    public Button button_der_resetFiltres_main;
    public AnchorPane rutes_pane;
    public TableView tableTop10Valorades;
    public TableColumn nomColumn;
    public TableColumn valueColumn;
    public Text textPrincipal;
    public ComboBox comboBox_main_grup;
    public ImageView image_main;
    public Button button1_abajo_main;
    public Button button2_abajo_main;
    public Button button3_abajo_main;
    @FXML
    public ComboBox comboBox_main_comarca;
    public ComboBox comboBox_main_localitat;

```

```
46      private String correuPersona;
47      private Integer id_grup;
48
49      private EscenaReservarAllotjament observable;
50
51      public void start() throws Exception {
52          this.controller = Controller.getInstance();
53          this.correuPersona = this.controller.getSessionMemory().getCorreuPersona();
54          asignarTextoPrincipal_Correo_y_Grupo(correuPersona);
55          asignarimagen();
56
57          popularComboBoxComarques(); //success
58          popularComboBoxLocalitat(); //success
59          popularRutesPerNom();
60
61      }
62      public void asignarTextoPrincipal_Correo_y_Grupo(String correuPersona) throws Exception {
63          //Paso1. Teniendo el correo de la persona, buscar el id del grupo.
64          ArrayList<HashMap<Object,Object>> resuListGrupHashMap= controller.getAllGrupsPerPersona(correuPersona);
65          //Paso2. Recoger el grupo por su id.
66          Integer firstidCB=null;
67          Integer lastidCB=null;
68          String firstnameCB=null;
69          Boolean flagGrup=false;
70
71          if(resuListGrupHashMap.size()>0){
72              firstidCB = (Integer) resuListGrupHashMap.get(0).get("id");
73              firstnameCB = (String) resuListGrupHashMap.get(0).get("nom");
74
75              for (HashMap<Object,Object> grup:resuListGrupHashMap){
76                  comboBox_main_grup.getItems().add(grup.get("nom"));
77                  lastidCB= (Integer) grup.get("id");
78                  flagGrup=true;
```

```
            }
        }else{
            firstnameCB="Empty Groups";
            comboBox_main_grup.setDisable(true);
        }
        if(flagGrup){
            this.controller.getSessionMemory().setIdGrup(lastidCB);
        }
        comboBox_main_grup.setPromptText(firstnameCB);
        textPrincipal.setText("TripUB: "+correuPersona);
        this.observable_comboBox_main_grup();
    }

    public void asignarimagen() throws FileNotFoundException {
        FileInputStream input = new FileInputStream("./src/main/view-resources/ub/edu/static-resources/logo8.png");
        Image image = new Image(input);
        image_main.setImage(image);
    }

    /*
    * Aquesta inner Class la associarem als items de la taula en el mètode popularTopActivitatsFetes
    * Seguir aquesta estructura per popular taules
    * */
    public static class DataTop {
        //Cal deixar aquests atributs com finals per poder popular la taula quan el mètode  la cridi
        private final SimpleStringProperty nom;
        private final SimpleStringProperty value;
        DataTop(String nom_param, Integer value_param){
            this.nom = new SimpleStringProperty(nom_param);
            this.value = new SimpleStringProperty(value_param.toString());
        }
        //métodos getters
        public String getNom() {
```

```
112             return nom.get();
113         }
114         public String getValue() {
115             return value.get();
116         }
117     }
118 
119     private void commonGroundOrderValoracio() {
120         nomColumn.setCellValueFactory(new PropertyValueFactory<DataTop, String>("nom"));
121         valueColumn.setCellValueFactory(new PropertyValueFactory<DataTop, String>("value"));
122         tableTop10Valorades.getItems().clear();
123     }
124 
125     private void loop(ArrayList<HashMap<Object,Object>> list) {
126         for(int i = 0; i < list.size(); i++) {
127             tableTop10Valorades.getItems().add(new DataTop(list.get(i).get("nom").toString(), (Integer)list.get(i).get("value")));
128         }
129     }
130 
131     public void popularTopXYZValorades() {
132         ArrayList<HashMap<Object,Object>> list = new ArrayList<>();
133         commonGroundOrderValoracio();
134         /* passem els hashmaps a una llista i no a un iterable */
135         for(HashMap<Object,Object> h : controller.getAllotjamentsPagaments()) list.add(h);
136 
137         loop(list);
138     }
139 
140     public void popularTopPerPagament(String tipusPagament) {
141         ArrayList<HashMap<Object,Object>> list = new ArrayList<>();
142         commonGroundOrderValoracio();
143         /* passem els hashmaps a una llista i no a un iterable */
144         for(HashMap<Object,Object> h : controller.getAllotjamentByPagament(tipusPagament)) list.add(h);
```

```
145
146         loop(list);
147     }
148
149     private void ajustaRutes(List<HashMap<Object,Object>> listaRutas) {
150         rutes_pane.getChildren().clear(); // hem d'eliminar les rutes perquè poden canviar totalment
151         List<Node> excursionsPaneChildren = rutes_pane.getChildren();
152         double width = ruta_btn.getWidth();
153         double height = ruta_btn.getHeight();
154         double layoutX = ruta_btn.getLayoutX();
155         double layoutY = ruta_btn.getLayoutY();
156
157         String text;
158         Button new_btn;
159         Integer id;
160         String nom;
161
162         for (HashMap<Object,Object> ruta:listaRutas) {
163             id = (Integer) ruta.get("id");
164             nom = (String) ruta.get("nom");
165
166             text = id.toString()+" | "+nom;
167             new_btn = createRutaButton(id, text, width, height, layoutX, layoutY);
168             excursionsPaneChildren.add(new_btn);
169             layoutY += ESPAI_ENTRE_BOTONS;
170         }
171
172         //Actualitzem la mida del pane que conté els botons perque es pugui fer scroll cap abaix si hi ha més botons dels que caben al pane
173         rutes_pane.setPrefHeight(layoutY);
174         //Esborrem excursio_btn, que l'utilitzavem únicament com a referència per la mida dels botons
175         excursionsPaneChildren.remove(ruta_btn);
176     }
177
```

```
178     private void popularRutesPerNom() {
179         List<HashMap<Object,Object>> listaRutas = controller.getAllRutesPerNom();
180         if(listaRutas == null || listaRutas.size()==0){
181             return;
182         }
183         ajustaRutes(listaRutas);
184     }
185 
186     /*
187     * Crea un botó de dimensions width x height, colocat a la posició (layoutX, layoutY)
188     * */
189     private Button createRutaButton(Integer id_ruta, String text, double width, double height, double layoutX, double layoutY){
190         Button new_btn = new Button();
191         new_btn.setPrefWidth(width);
192         new_btn.setPrefHeight(height);
193         new_btn.setText(text);
194         new_btn.setLayoutX(layoutX);
195         new_btn.setLayoutY(layoutY);
196         new_btn.setAlignment(Pos.BASELINE_LEFT);
197         //asignamos el evento del click que ejecutará mostrar la ventana con detalles de la ruta
198         new_btn.setOnMouseClicked(event ->
199         {
200             if (event.getButton() == MouseButton.PRIMARY)
201             {
202                 try {
203                     mostrarFinestraRuta(id_ruta);
204                 } catch (Exception e) {
205                     e.printStackTrace();
206                 }
207             }
208         });
209         return new_btn;
210     }
```

```
211
212     private void mostrarFinestraRuta(Integer id) throws Exception {
213         //Nova finestra
214         Escena escena = EscenaFactory.INSTANCE.creaEscena("rutaDetalls-view", "Detalls ruta "+String.valueOf(id));
215         EscenaRutaDetalls escenaRutaDetalls = ((EscenaRutaDetalls)escena);
216         escenaRutaDetalls.setObserver(this);
217         escenaRutaDetalls.setController(controller);
218         this.controller.getSessionMemory().setIdRuta(id);
219         escenaRutaDetalls.start();
220     }
221
222     public void observable_comboBox_main_grup(){
223         comboBox_main_grup.valueProperty().addListener(new ChangeListener<String>() {
224             @Override public void changed(ObservableValue classObject, String oldValue, String newValue) {
225                 Integer idGrup = controller.getIdGrupByName(newValue);
226                 controller.getSessionMemory().setIdGrup(idGrup);
227                 System.out.println("Filtrar contenido por grup: id: "+idGrup+" name: "+newValue);            }
228         });
229     }
230
231     public void popularComboBoxComarques() throws Exception {
232         List<HashMap<Object,Object>> comarques = controller.getAllComarques();
233         System.out.println("comarques: "+comarques);
234         String s = comboBox_main_comarca.getPromptText();
235         comboBox_main_comarca.getItems().add(0,s);
236
237         Integer id=null;
238         String nom=null;
239         for (HashMap<Object,Object> comarca: comarques) {
240             if(comarca.get("id")!=null){id=(Integer) comarca.get("id");}
241             if(comarca.get("nom")!=null){nom=(String) comarca.get("nom");}
242             //el index del comboBox debe empezar por 0,
243             //pero mas arriba en el 0 hemos añadido el default
```

```
244             //el que hará de reset del filtro
245             //IMPORTANTE: la lista debe estar ordenada por sus ids (ver como se ordenda para popularComboBoxLocalitat)
246             //comboBox_main_comarca.getItems().add(id,nom);
247             //si pasamos del id, no nos petará en caso de que no esté ordenada
248             comboBox_main_comarca.getItems().add(nom);
249         }
250 
251         //añadir el listener del combobox
252         comboBox_main_comarca.valueProperty().addListener(new ChangeListener<String>() {
253             //OPCIÓN-1 -> asignar listener para que se ejecute cuando detecte el cambio
254             @Override public void changed(ObservableValue classObject, String oldValue, String newValue) {
255                 // TODO: extensión de popular la lista de Rutas
256                 List<HashMap<Object,Object>> listaRutas = controller.llistarRutesPerComarques(newValue);
257                 if(listaRutas == null || listaRutas.size()==0) {
258                     HashMap<Object,Object> e = new HashMap<>();
259                     e.put("id",0);
260                     e.put("nom","No hi ha cap ruta en aquesta comarca");
261                     listaRutas.add(e);
262                 }
263                 ajustaRutes(listaRutas);
264 
265                 if(newValue.equals(comboBox_main_comarca.getPromptText())){
266                     popularRutesPerNom();
267                 }
268             }
269         });
270     }
271 
272     public void popularComboBoxLocalitat() throws Exception {
273         List<HashMap<Object,Object>> localitats = controller.getAllLocalitats();
274         System.out.println("localitats: "+localitats);
275         String s = comboBox_main_localitat.getPromptText();
276         comboBox_main_localitat.getItems().add(0,s);
```

```
277
278         Integer id=null;
279         String nom=null;
280         for (HashMap<Object,Object> loc: localitats) {
281             if(loc.get("id")!=null){id=(Integer) loc.get("id");}
282             if(loc.get("nom")!=null){nom=(String) loc.get("nom");}
283             //el index del comboBox debe empezar por 0
284             //pero mas arriba en el 0 hemos añadido el default
285             //el que hará de reset del filtro
286             // IMPORTANTE: la lista debe estar ordenada por sus ids
287             //comboBox_main_localitat.getItems().add(id,nom);
288             //si pasamos del id, no nos petará en caso de que no esté ordenada
289             comboBox_main_localitat.getItems().add(nom);
290         }
291
292         //añadir el listener del combobox
293         comboBox_main_localitat.valueProperty().addListener(new ChangeListener<String>() {
294             //OPCIÓN-1 -> asignar listener para que se ejecute cuando detecte el cambio
295             @Override public void changed(ObservableValue classObject, String oldValue, String newValue) {
296                 //TODO: extensión de popular la lista de Rutas
297                 List<HashMap<Object,Object>> listaRutas = controller.llistarRutesPerLocalitat(newValue);
298
299                 if(listaRutas == null || listaRutas.size()==0) {
300                     HashMap<Object,Object> e = new HashMap<>();
301                     e.put("id",0);
302                     e.put("nom","No hi ha cap ruta en aquesta localitat");
303                     listaRutas.add(e);
304                 }
305                 ajustaRutes(listaRutas);
306
307                 if(newValue.equals(comboBox_main_localitat.getPromptText())){
308                     popularRutesPerNom();
309                 }
```

```
310             }
311         });
312     }
313     public void onButtonIzqResetFiltresClick(){
314         this.popularRutesPerNom();
315 
316         //reset filtro ComboBox1 Comarcas
317         //Object stringNameOfComboBoxComarcas = comboBox_main_comarca.getValue();
318         //System.out.println("stringNameOfComboBoxComarcas:"+stringNameOfComboBoxComarcas);
319         //actualizar el combobox por ese stringValue
320         Object str_default_ComboBox_comarcas = comboBox_main_comarca.getPromptText();
321         //System.out.println("str_default_ComboBox_comarcas:"+str_default_ComboBox_comarcas);
322         comboBox_main_comarca.setValue(str_default_ComboBox_comarcas);
323 
324         //reset filtro ComboBox2 Localitat
325         //Object stringNameOfComboBoxLocalitat = comboBox_main_localitat.getValue();
326         //System.out.println("stringNameOfComboBoxLocalitat:"+stringNameOfComboBoxLocalitat);
327         //actualizar el combobox por ese stringValue
328         Object str_default_ComboBox_localitat = comboBox_main_localitat.getPromptText();
329         //System.out.println("str_default_ComboBox_localitat:"+str_default_ComboBox_localitat);
330         comboBox_main_localitat.setValue(str_default_ComboBox_localitat);
331     }
332 
333     public void onButtonDerResetFiltresClick(){
334         //OPCIÓN-2 -> asignar una funcion asociada al click del botón
335         System.out.println("Hacer reset filtros derecha en EscenaMain");
336         this.popularTopXYZValorades();
337     }
338 
339     public void onButton1Click(){
340         System.out.println("Filtrem en funció del criteri, boton1");
341         this.popularTopPerPagament(button1_abajo_main.getText());
342     }
```

```
343     public void onButton2Click(){
344         System.out.println("Filtrem en funció del criteri, boton2");
345         this.popularTopPerPagament(button2_abajo_main.getText());
346     }
347     public void onButton3Click() {
348         System.out.println("Filtrem en funció del criteri, boton3");
349         this.popularTopPerPagament(button3_abajo_main.getText());
350     }
351 
352     // OBSERVER
353     @Override
354     public void setSubject(Escena escena) {
355         this.observable = (EscenaReservarAllotjament) escena;
356     }
357     @Override
358     public void update(){
359         this.popularTopXYZValorades();
360     }
361 }
```

Listing 3: Classe *EscenaMain* del nostre projecte.

```
1  package ub.edu.view;
2  
3  import javafx.scene.control.Button;
4  import javafx.scene.control.RadioButton;
5  import javafx.scene.control.ToggleGroup;
6  import javafx.scene.text.Text;
7  
8  import java.lang.reflect.Array;
9  import java.util.ArrayList;
10 
```

```
11  public class EscenaReservarAllotjament extends Escena implements SceneObservable {
12      public Text nomAllotjamant_text;
13      public Text preuPerNitAllotjament_text;
14      public Text preuEsmorzar_text;
15      public Text preuMP_text;
16      public Text preuPC_text;
17      public RadioButton radioButton_Group1_Text1;
18      public RadioButton radioButton_Group1_Text2;
19      public RadioButton radioButton_Group1_Text3;
20      public Button button_reservar;
21      public Button button_cancel;
22      private String correu_persona;
23      private Integer id_ruta;
24      private Integer id_tram_etapa;
25      private Integer id_allotjament;
26
27      // aplicarem el patró observer: volem que EscenaMain estigui subscrita als canvis d'EscenaReservarAllotjament
28      private EscenaMain subscriber;
29
30
31      public void start() throws Exception {
32          System.out.println("Entro en Reservar Allotjament");
33          this.correu_persona = this.controller.getSessionMemory().getCorreuPersona();
34          this.id_ruta = this.controller.getSessionMemory().getIdRuta();
35          this.id_tram_etapa = this.controller.getSessionMemory().getIdTram();
36          this.id_allotjament = this.controller.getSessionMemory().getIdAllotjament();
37          initialize_RB();
38      }
39
40      private void initialize_RB() {
41          ToggleGroup group = new ToggleGroup();
42          radioButton_Group1_Text1.setToggleGroup(group);
43          radioButton_Group1_Text2.setToggleGroup(group);
```

```
44          radioButton_Group1_Text3.setToggleGroup(group);
45      }
46
47      public void onButtonReservarClick() {
48          //enviar la reserva
49          System.out.println("TODO: Entro a enviar una reserva");
50
51          //TODO:
52          // La idea es: guardar la reserva en el modelo y actualizar la vista en caso necesario
53          // Para ello:
54          // 1-Recoger los datos seleccionados de la vista (cómo se recogen? ver código de más abajo)
55          // 2-Conectar con el controller pasandole los parametros necesarios para que
56          // el controller el modelo (y opcionalmente se ejecute el metodo add de los respectivos DAO_Reserva_DB)
57          // 3- Refrescar la vista si es necesario
58
59          // És un Open-Closed com una casa, però no veiem alternativa...
60          String typeReserva;
61
62          if (radioButton_Group1_Text1.isSelected()) {
63              typeReserva = "ABANS_AMB_ANULACIO";
64          }
65          else if(radioButton_Group1_Text2.isSelected()) {
66              typeReserva = "ABANS_SENSE_ANULACIO";
67          }
68          else {
69              typeReserva = "EN_ARRIBAR";
70          }
71
72          System.out.println("Reserva de tipus: " + typeReserva);
73          controller.reservarAllotjament(id_ruta, id_tram_etapa, id_allotjament, correu_persona, typeReserva, 3);
74
75          // refresquem la vista //
76          this.subscriber.update();
```

```
77
78          stage.close();
79      }
80
81      public void onButtonCancelarClick(){
82          //cancelar la reserva
83          System.out.println("Entro en cancelar una reserva");
84          controller.anularReserva(id_allotjament,correu_persona);
85
86          // refresquem la vista //
87          this.subscriber.update();
88
89          stage.close();
90      }
91      @Override
92      public void setObserver(EscenaMain observer) {
93          this.subscriber = observer;
94      }
95  }
96
97
```

Listing 4: Classe *EscenaReservarAllotjament* del nostre projecte.

```
1   package ub.edu.resources;
2
3   import ub.edu.model.StatusType;
4   import ub.edu.model.Seguretat;
5   import ub.edu.model.*;
6   import ub.edu.resources.dao.Parell;
7   import ub.edu.resources.service.AbstractFactoryData;
8   import ub.edu.resources.service.DataService;
9
```

```
10  import java.util.List;
11
12  import ub.edu.resources.service.FactoryDB;
13
14  public class ResourcesFacade {
15
16      private AbstractFactoryData factory;      // Origen de les dades
17      private DataService dataService;          // Connexio amb les dades
18      private TripUB tripUB;                    // Model a tractar
19
20      // private ModelFacade modelFacade;      // desacoblem i no utilitzem
21
22      private TramFacade tramFacade;
23      private AllotjamentFacade allotjamentFacade;
24      private PersonFacade personFacade;
25      private LocalitatsFacade localitatsFacade;
26      private RutaFacade rutaFacade;
27
28      private static ResourcesFacade resourcesFacade;
29
30      private ResourcesFacade(TripUB tripUB, TramFacade tramFacade,
31                              AllotjamentFacade allotjamentFacade, PersonFacade personFacade,
32                              LocalitatsFacade localitatsFacade, RutaFacade rutaFacade) {
33          factory = FactoryDB.getInstance();
34          dataService = DataService.getInstance(factory);
35          this.tripUB = tripUB;
36          this.tramFacade = tramFacade;
37          this.allotjamentFacade = allotjamentFacade;
38          this.personFacade = personFacade;
39          this.localitatsFacade = localitatsFacade;
40          this.rutaFacade = rutaFacade;
41      }
42
```

```
43      public static ResourcesFacade getInstance(TripUB tripUB, TramFacade tramFacade,
44                                                AllotjamentFacade allotjamentFacade, PersonFacade personFacade,
45                                                LocalitatsFacade localitatsFacade, RutaFacade rutaFacade){
46          if(resourcesFacade == null) {
47              resourcesFacade = new ResourcesFacade(tripUB,tramFacade,allotjamentFacade,personFacade,localitatsFacade,rutaFacade);
48          }
49          return resourcesFacade;
50      }
51
52      public void populateTripUB() {
53          try {
54              initXarxaPersones(); //success
55              initGrups(); //success
56              relacioPersonesGrups(); //success
57
58              initRutes(); //success
59              initComarques(); //success
60              relacionarRutesComarques(); //success
61              initLocalitats();  //success // Pressuposa que les comarques ja han estat creades
62
63              initTramsRuta();  //success // Pressuposa que les rutes ja estan creades.
64              initAllotjaments(); //success // Pressuposa que les etapes ja han estat creades
65
66              initPerfilsPersones(); //success
67
68              initAllotjamentsList();
69              initReservesList();
70
71          } catch (Exception e) {
72              System.out.println("Exception: --> " + e.getMessage());
73          }
74      }
75
```

```
76      private void relacioPersonesGrups() throws Exception {
77          // Han estat creats els grups i les persones previament
78          // Ara cal lligar les persones i els grups
79          List<Parell<String, Integer>> relacionsPG = dataService.getAllGrupFormatPersones();
80
81          for (Parell<String, Integer> parell: relacionsPG) {
82              Persona persona;
83              persona = personFacade.findPersonaByCorreu(parell.getElement1());
84              Grup grup;
85              grup = tripUB.findGrup(parell.getElement2());
86              grup.addPersona(persona);
87
88              PerfilPersona pp = persona.getPerfil();
89              pp.addGrup(grup);
90          }
91
92      }
93
94      private void initPerfilsPersones() throws Exception {
95          // Es considera que totes les classes amb les que estan relacionades
96          // vots, opinio i reserves ja han estat creades
97          initVots();
98          initOpinio();
99          initReserves();
100     }
101
102     private void initOpinio() throws Exception  {
103         List<Opinio> llista = dataService.getAllOpinions();
104         for (Opinio o: llista) {
105             o.setPuntDePas(localitatsFacade.findPuntDePas(o.getIdPuntDePas()));
106             Persona persona = personFacade.findPersonaByCorreu(o.getCorreuPersona());
107             o.setPersona(persona);
108
```

```
109             PerfilPersona pp = persona.getPerfil();
110             pp.addOpinio(o);
111         }
112     }
113 
114     private void initVots() throws Exception  {
115         List<Vot> llista = dataService.getAllVots();
116         for (Vot v: llista) {
117             v.setRuta(tripUB.findRuta((v.getIdRuta())));
118             Persona persona = personFacade.findPersonaByCorreu(v.getCorreuPersona());
119             v.setPersona(persona);
120             v.setGrup(tripUB.findGrup(v.getIdGrup()));
121 
122             PerfilPersona pp = persona.getPerfil();
123             pp.addVot(v);
124         }
125     }
126 
127     private void initReserves() throws Exception  {
128         List<Reserva> llista = dataService.getAllReservas();
129         for (Reserva r: llista) {
130             Persona persona = personFacade.findPersonaByCorreu(r.getCorreuPersona());
131 
132             PerfilPersona pp = persona.getPerfil();
133             pp.addReserva(r);
134         }
135     }
136     private void initLocalitats() throws Exception  {
137         List<Localitat> llista = dataService.getAllLocalitats();
138         for (Localitat l : llista) {
139             Comarca c = tripUB.findComarca(l.getIdComarca());
140             l.setIdComarca(c.getId());
141             c.addLocalitat(l);
```

```
            List<Ruta> origens = tripUB.findRutaByOrigen(l.getId());
            for (Ruta r: origens) {
                r.setLocalitatOrigen(l);
            }
            List<Ruta> destins = tripUB.findRutaByDesti(l.getId());
            for (Ruta r: destins) {
                r.setLocalitatDesti(l);
            }
        }
    }

    private void initAllotjaments() throws Exception {
        List<Allotjament> lallotjaments= dataService.getAllAllotjaments();
        for (Allotjament a : lallotjaments) {
            //Etapa e = dataService.getEtapaById(a.getIdEtapa());
            Etapa e = localitatsFacade.findEtapa(a.getIdEtapa());
            if (e!=null) {
                e.addAllotjament(a);
                a.setIdEtapa(e.getId());
            }
        }
    }

    private void initTramsRuta() throws Exception {
        // Es donen d'alta els trams d'una ruta i depres, segons siguin
        // de tipus etapa o de tipus track es donen d'alta
        // les etapes o els punts de control
        // Al final es reparteixen els transports pels trams ja creats
        // no es consulta al dataservice pels trams sino a tripUB per
        // a no fer dos vegades new del mateix tram

        initTramsEtapaRuta();
        initTramsTrackRuta();
```

```
175         // Lligar transports amb els trams etapa. Els transports també es creen a demanda
176         // segons el que necessiti el tram.
177         List<Parell<Integer, Integer>> llistaTransportsTrams= dataService.getAllTransportsPosibleTrams();
178 
179         for (Parell p: llistaTransportsTrams) {
180             // element 1 Transport
181             // element 2 tram
182             Transport t = dataService.getTransportById((Integer)(p.getElement1()));
183             tramFacade.addTransportTram((Integer)(p.getElement2()), t);
184         }
185     }
186 
187     private void initTramsEtapaRuta() throws Exception {
188         // Es donen d'alta els trams d'una ruta i depres, segons siguin
189         // de tipus etapa es donen d'alta
190         // les etapes
191         // Al final es reparteixen els transports pels trams ja creats
192         // no es consulta al dataservice pels trams sino a tripUB per
193         // a no fer dos vegades new del mateix tram
194 
195         List<TramEtapa> trams = dataService.getAllTramEtapas();
196         for (TramEtapa te : trams) {
197             // Reparticio dels trams a les rutes
198             tramFacade.addTram(te);
199 
200             // Lligar etapa-TramEtapa. Les etapes es llegeixen una a una de la BD
201             // segons ho necessiti el tram etapa
202 
203             Etapa etapa = dataService.getEtapaById(te.getIdEtapaDesti());
204             te.setEtapaDesti(etapa);
205             etapa = dataService.getEtapaById(te.getIdEtapaOrigen());
206             te.setEtapaOrigen(etapa);
207         }
```

```
208     }
209     private void initTramsTrackRuta() throws Exception {
210             // Es donen d'alta els trams d'una ruta i depres, segons siguin
211             // d de tipus track es donen d'alta
212             // els punts de control
213
214             List<TramTrack> trams = dataService.getAllTramTracks();
215             for (TramTrack tt: trams) {
216                 // Reparticio dels trams a les rutes
217                 tramFacade.addTram(tt);
218
219                 PuntDePas pdp = dataService.getPuntDePasById(tt.getIdPuntDePasDesti());
220                 tt.setPuntDePasDesti(pdp);
221                 pdp = dataService.getPuntDePasById(tt.getIdPuntDePasOrigen());
222                 tt.setPuntDePasOrigen(pdp);
223             }
224         }
225
226
227     public boolean initGrups() throws Exception {
228         return (tripUB.setGrups(dataService.getAllGrups()));
229     }
230
231     public boolean initXarxaPersones() throws Exception {
232         return personFacade.setXarxaPersones(dataService.getAllPersonas());
233     }
234
235     public boolean initRutes() throws Exception {
236         List<Ruta> listRutas = dataService.getAllRutas();
237         return tripUB.setRutes(listRutas);
238     }
239
240     public boolean initComarques() throws Exception {
```

```
241         List<Comarca> listComarques = dataService.getAllComarcas();
242         return tripUB.setComarques(listComarques);
243     }
244 
245     public boolean initAllotjamentsList() throws Exception {
246         List<Allotjament> listAllotjaments = dataService.getAllAllotjaments();
247         return tripUB.setAllotjaments(listAllotjaments);
248     }
249 
250     public boolean initReservesList() throws Exception{
251         List<Reserva> listReserva = dataService.getAllReservas();
252         return tripUB.setReserves(listReserva);
253     }
254     public void relacionarRutesComarques() throws Exception {
255         //DB:
256         List<Parell<Integer, Integer>> relacionsRC = dataService.getAllRelacioComarcasRutas();
257         tripUB.setComarquesToRutes(relacionsRC);
258     }
259 
260 
261     public StatusType addNewPersona(String correu, String nom, String cognoms, String dni, String password, Integer id_tripUB) throws Exception
262         if  (personFacade.findPersonaByCorreu(correu) != null)
263             return StatusType.PERSONA_DUPLICADA;
264         else if (Seguretat.isMail(correu) && Seguretat.isPasswordSegur(password)){
265             Persona persona = new Persona(correu, nom, cognoms, dni, password, id_tripUB);
266             if(this.dataService.addPersona(persona)){
267                 //actualizar lista
268                 this.initXarxaPersones();
269             }
270             return StatusType.REGISTRE_VALID;
271         }
272         else return StatusType.FORMAT_INCORRECTE_CORREU_PWD;
273     }
```

```
274
275
276     public Integer addNewGrup(String grupNom) throws Exception {
277         Grup grup = new Grup(grupNom);
278         dataService.addGrup(grup);
279         tripUB.addGrup(grup);
280         return grup.getId();
281     }
282     public boolean addRelacionGrupPersona(String correuPersona, Integer idGrup) throws Exception {
283         Parell<String,Integer> gfp = new Parell<>(correuPersona,idGrup);
284         personFacade.addPersonaToGrup(idGrup, correuPersona);
285         return dataService.addGrupFormatPersones(gfp);
286     }
287
288     public void addReserva(Reserva reserva) throws Exception {
289         if(reserva == null) throw new Exception("Aquesta reserva és nul·la");
290         if(this.dataService.getAllReservas().contains(reserva)) throw new Exception("Aquesta reserva ja ha estat afegida");
291         boolean bool = this.dataService.addReserva(reserva);
292         if(!bool) throw new Exception("Aquesta reserva no s'ha pogut afegir, per raons desconegudes");
293     }
294
295     public void removeReserva(Reserva reserva) throws Exception {
296         if(reserva == null) throw new Exception("Aquesta reserva és nul·la");
297         boolean b = false;
298         for(Reserva iter : dataService.getAllReservas()) { if(iter.getId().equals(reserva.getId())) { b=true; break; } }
299         if(!b) throw new Exception("No hem pogut trobar la teva reserva");
300         boolean bool = this.dataService.deleteReserva(reserva);
301         if(!bool) throw new Exception("Aquesta reserva no s'ha pogut eliminar, per raons desconegudes");
302     }
303
304 }
```

Listing 5: Classe *ResourcesFacade* del nostre projecte.

```java
package ub.edu.model;

import java.time.LocalDate;
import java.util.*;

public class AllotjamentFacade {

    private TripUB tripUB;

    public static AllotjamentFacade allotjamentFacade;

    private AllotjamentFacade(TripUB tripUB) {
        this.tripUB = tripUB;
    }

    public static AllotjamentFacade getInstance(TripUB tripUB) {
        if(allotjamentFacade == null){
            allotjamentFacade = new AllotjamentFacade(tripUB);
        }
        return allotjamentFacade;
    }

    public List<HashMap<Object, Object>> getAllAllotjaments() throws Exception {
        List<Allotjament> allotjamentList = getAllAllotjamentsUB();
        List<HashMap<Object, Object>> listaFinal = new ArrayList<>();
        for (Allotjament allotjament : allotjamentList) {
            HashMap<Object, Object> hashMap = new HashMap<>();
            hashMap.put("id", allotjament.getId());
            hashMap.put("nom", allotjament.getNom());
            listaFinal.add(hashMap);
        }
        return listaFinal;
    }
```

```
34
35      public List<Allotjament> getAllAllotjamentsUB() {
36          List<Allotjament> listAllotjaments = new ArrayList<>();
37          for (Ruta r: tripUB.getRutes()) {
38              List<Allotjament> le = r.getAllAllotjaments();
39              for (Allotjament te: le) {
40                  listAllotjaments.add(te);
41              }
42          }
43          return listAllotjaments;
44      }
45
46      public Allotjament findAllotjament(Integer idAllotjament) {
47          Allotjament a = null;
48          for (Ruta r: tripUB.getRutes()) {
49              a = r.findAllotjament(idAllotjament);
50              if (a != null) {
51                  return a;
52              }
53          }
54          return a;
55      }
56
57      private Reserva checkReserva(Integer nomAllotjament, String nomPersona) throws Exception
58      {
59          LocalDate dema = LocalDate.now().plusDays(1);
60
61          boolean allowReserva;
62          boolean diesCorrectes;
63
64          // en cas que la persona existeixi iterem sobre les reserves que ha fet per a veure si alguna coincideix amb la que volem eliminar
65          Persona persona = tripUB.getXarxaPersones().find(nomPersona);
66          if(persona == null) throw new Exception("Aquesta persona no existeix");
```

```
67          PerfilPersona perfil = persona.getPerfil();
68          for(Reserva reserva: perfil.getAllReserves().values()) {
69
70              // si trobem la reserva
71              if(reserva.getIdAllotjament().equals(nomAllotjament)) {
72
73                  // es pot anul·lar
74                  allowReserva = reserva.getAnulacio();
75
76                  // reserva després de 24h
77                  diesCorrectes = reserva.getDateCheckIn().isBefore(dema);
78
79                  // en cas que no es pugui fer una anul·lació, sigui per no permetre-ho o perquè falta menys d'un dia
80                  if(!(allowReserva && diesCorrectes)){
81                      throw new Exception("Aquesta reserva no es pot anul·lar");
82                  }
83                  return reserva;
84              }
85          }
86          throw new Exception("La persona no ha fet aquesta reserva");
87      }
88
89
90      public Reserva crearReserva(Integer idRuta, Integer idTram, Integer idAllotjament, String correuPersona,
91                                  String tipusPagament, int numNits) throws Exception
92      {
93
94          /* mirem que les dades siguin correctes */
95          if(tripUB.getRutes().isEmpty()) throw new Exception("La llista de rutes està buida");
96          if(tripUB.getAllTrams().isEmpty()) throw new Exception("La llista de trams està buida");
97
98          Persona persona = tripUB.findPersona(correuPersona);
99          if(persona == null) throw new Exception("La persona no s'ha trobat.");
```

```
100
101         // obtenim l'allotjament amb el nomAllotjament que passem per paràmetre
102         Allotjament allotjament = tripUB.getAllotjament(idAllotjament);
103         if(allotjament == null) throw new Exception("Aquest allotjament no existeix");
104
105         // guardem en una variable la ruta que estem fent
106         Ruta ruta = tripUB.getRuta(idRuta);
107         if(ruta == null) throw new Exception("La ruta que ens has proporcionat no existeix");
108
109         // Tram tram = ruta.findTramEtapaById(idTram);
110
111         LocalDate dataFinal = ruta.getDataCreacio().plusDays(numNits);
112         Reserva reserva = new Reserva(correuPersona, idRuta, idTram, ruta.getDataCreacio(), dataFinal, idAllotjament, tipusPagament);
113
114         return reserva;
115     }
116
117     public String reservarAllotjament(Reserva reserva) throws Exception {
118         tripUB.addReservaToPersona(reserva.getCorreuPersona(),reserva);
119         tripUB.addReserva(reserva);
120
121         return "Hem afegit la teva reserva correctament.";
122     }
123
124     public Reserva anularReserva(Integer nomAllotjament, String nomPersona) throws Exception
125     {
126         /* mirem que l'usuari hagi fet la reserva */
127         Reserva reserva = checkReserva(nomAllotjament, nomPersona);
128
129         tripUB.removeReserva(reserva);
130         tripUB.removeReservaFromPersona(nomPersona,reserva);
131
132         return reserva;
```

```
133     }
134
135     public Iterable<HashMap<Object,Object>> getAllotjamentsPagaments() {
136         /* declaració d'algunes variables interessants en la resolució d'aquest mètode */
137         List <List<Reserva>> reserves = new ArrayList<>();
138         List <HashMap<Object,Object>> fi = new ArrayList<>();
139         HashMap<Object,Object> h;
140
141         if(tripUB.getAllAllotjaments().isEmpty()) {
142             h = new HashMap<>();
143             h.put("nom","No tenim allotjaments");
144             h.put("value",0);
145             h.put("tipusPagament",null);
146             fi.add(h);
147             return fi;
148         }
149
150         /* ens quedem amb les reserves d'aquests allotjaments */
151         for(Allotjament allotjament : tripUB.getAllAllotjaments()) {
152             List <Reserva> res = tripUB.getReservesByAllotjament(allotjament.getId());
153             reserves.add(res);
154         }
155
156         /* ens quedem amb els allotjaments i ens quedem amb els que té més nombre d'ocurrències */
157         reserves.sort(new SortBySize2());
158         int max = reserves.get(reserves.size()-1).size();
159
160         /* ens demanen obtenir el top 5 */
161         while(reserves.size() > 5) {
162             if(reserves.get(0).size() == max) break;
163             else reserves.remove(0);
164         }
165
```

```
166         /* iterem per la llista de llistes per trobar el tipus de pagament que s'utilitza més */
167         List<String> ocurrencies = new ArrayList<>(); int curr = 0; int high = 0; String key = null;
168         for(int i = 0; i < reserves.size(); ++i) {
169             /* afegim tots els tipus de pagament en una llista i ens quedem amb el més recurrent */
170             List<Reserva> res = reserves.get(i);
171             for(int j = 0; j < res.size(); ++j) {
172                 ocurrencies.add(res.get(j).getTipusPagament());
173             }
174             for (String iter : ocurrencies) {
175                 curr = Collections.frequency(ocurrencies, iter);
176                 if(high < curr) {
177                     high = curr;
178                     key = iter;
179                 }
180             }
181             if(!(res.isEmpty())) {
182                 h = new HashMap<>();
183                 /* hem de fer un workaround per obtenir l'allotjament, ja que no el detecta directament amb l'atribut */
184                 /* igualment, això ens beneficiarà de cara a desacoblar el codi més endavant */
185                 h.put("nom",findAllotjament(res.get(0).getIdAllotjament()).getNom() + ", " + key);
186                 h.put("tipusPagament",key);
187                 h.put("value",high);
188                 fi.add(h);
189             }
190             ocurrencies = new ArrayList<>();
191             high = 0; curr = 0; key = null;
192         }
193         if(fi.isEmpty()) {
194             h = new HashMap<>();
195             h.put("nom","No tenim reserves");
196             h.put("tipusPagament",null);
197             h.put("value",0);
198             fi.add(h);
```

```
199         }
200         return fi;
201     }
202 
203     public Iterable<HashMap<Object,Object>> getAllotjamentByPagament(String tipusPagament) {
204         /* declaració d'algunes variables interessants en la resolució d'aquest mètode */
205         List <List<Reserva>> reserves = new ArrayList<>();
206         List <HashMap<Object,Object>> fi = new ArrayList<>();
207         HashMap<Object,Object> h;
208 
209         if(tripUB.getAllAllotjaments().isEmpty()) {
210             h = new HashMap<>();
211             h.put("nom","No tenim allotjaments");
212             h.put("value",0);
213             h.put("tipusPagament",null);
214             fi.add(h);
215             return fi;
216         }
217 
218         /* ens quedem amb les reserves d'aquests allotjaments */
219         for(Allotjament allotjament : tripUB.getAllAllotjaments()) {
220             List <Reserva> res = tripUB.getReservesByAllotjament(allotjament.getId());
221             if(res.isEmpty()) continue; /* passem directament a la següent iteració */
222             /* eliminem aquelles que no tinguin el tipus de pagament que volem, evitem ConcurrentModification */
223             for(Reserva reserva : tripUB.getReservesByAllotjament(allotjament.getId())) {
224                 /* si la reserva no és del tipus de pagament que volem, ens la carreguen */
225                 if(!(reserva.getTipusPagament().equals(tipusPagament))) res.remove(reserva);
226             }
227             reserves.add(res);
228         }
229 
230         /* ens quedem amb els allotjaments i ens quedem amb els que té més nombre d'ocurrències */
231         reserves.sort(new SortBySize2());
```

```
232         int max = reserves.get(reserves.size()-1).size();
233
234         /* ens demanen obtenir el top 5 */
235         while(reserves.size() > 5) {
236             if(reserves.get(0).size() == max) break;
237             else reserves.remove(0);
238         }
239
240         /* iterem per la llista de llistes per trobar el tipus de pagament que s'utilitza més */
241         for(int i = 0; i < reserves.size(); ++i) {
242             List<Reserva> res = reserves.get(i);
243             /* semblant al que ja hem programat abans */
244             if(!(res.isEmpty())) {
245                 h = new HashMap<>();
246                 h.put("nom",findAllotjament(res.get(0).getIdAllotjament()).getNom());
247                 h.put("tipusPagament",tipusPagament);
248                 h.put("value",res.size());
249                 fi.add(h);
250             }
251         }
252         if(fi.isEmpty()) {
253             h = new HashMap<>();
254             h.put("nom","No tenim reserves");
255             h.put("tipusPagament",null);
256             h.put("value",0);
257             fi.add(h);
258         }
259         return fi;
260     }
261 }
262
263 class SortBySize2 implements Comparator<List<Reserva>> {
264     @Override
```

```
265     public int compare(List a, List b) {
266         return a.size()-b.size();
267     }
268 }
```

Listing 6: Classe *AllotjamentFacade* del nostre projecte.

```
1  package ub.edu.model;
2
3  import java.util.*;
4
5  public class LocalitatsFacade {
6      private TripUB tripUB;
7      public static LocalitatsFacade localitatsFacade;
8
9      private LocalitatsFacade(TripUB tripUB) {
10         this.tripUB = tripUB;
11     }
12
13     public static LocalitatsFacade getInstance(TripUB tripUB) {
14         if(localitatsFacade == null){
15             localitatsFacade = new LocalitatsFacade(tripUB);
16         }
17         return localitatsFacade;
18     }
19
20     public List<HashMap<Object, Object>> getAllComarques() throws Exception {
21         List<Comarca> l = tripUB.getAllComarques();
22         List<HashMap<Object, Object>> comarquesDisponibles = new ArrayList<>();
23         for (Comarca c : l) {
24             HashMap<Object, Object> hashMap = new HashMap<>();
25             hashMap.put("id", c.getId());
```

```
            hashMap.put("nom", c.getNom());
            comarquesDisponibles.add(hashMap);
        }
        return comarquesDisponibles;
    }

    public List<HashMap<Object, Object>> getAllLocalitats() throws Exception {
        List<Localitat> locs = getAllLocalitatsUB();
        List<HashMap<Object, Object>> localitats = new ArrayList<>();
        for (Localitat l : locs) {
            HashMap<Object, Object> hashMap = new HashMap<>();
            hashMap.put("id", l.getId());
            hashMap.put("nom", l.getNom());
            hashMap.put("altitud", l.getAltitud());
            hashMap.put("latitud", l.getLatitud());
            hashMap.put("longitud", l.getLongitud());
            hashMap.put("id_comarca", l.getIdComarca());
            localitats.add(hashMap);
        }
        return localitats;
    }

    public List<Localitat> getAllLocalitatsUB() {
        List<Localitat> llistaTotal = new ArrayList<>();
        for (Comarca c: tripUB.getAllComarques()) {
            List<Localitat>  l;
            l = c.getAllLocalitats();
            for (Localitat local: l) {
                llistaTotal.add(local);
            }
        }
        //la lista resultante no esta ordenada, mejor ordenarla ahora,
        //porque despeus en la vista si coges por id y esta desordenada, petará
```

```
        Collections.sort(llistaTotal, new Comparator<Localitat>() {
            @Override
            public int compare(Localitat l1, Localitat l2) {
                // Aqui esta el truco, ahora comparamos p2 con p1 y no al reves como antes
                return l1.getId().compareTo(l2.getId());
            }
        });
        return llistaTotal;
    }

    public HashMap<Object, Object> getLocalitat_y_PuntDePasById(Integer id) throws Exception {
        PuntDePas puntDePas = getPuntDePasByIdUB(id);
        Localitat localitat = getLocalitatByIdUB(puntDePas.getId());
        //al ser herencia recordar que id_localitat == id_puntDePas
        HashMap<Object, Object> hashMap = new HashMap<>();
        hashMap.put("id", puntDePas.getId());
        hashMap.put("nom", localitat.getNom());
        hashMap.put("highlight", puntDePas.getHighlight());
        return hashMap;
    }

    public HashMap<Object, Object> getLocalitatByID(Integer id) throws Exception {
        Localitat loc = getLocalitatByIdUB(id);
        HashMap<Object,Object> hashMap = new HashMap();
        if (loc != null) {
            hashMap.put("id", loc.getId());
            hashMap.put("nom", loc.getNom());
            hashMap.put("altitud", loc.getAltitud());
            hashMap.put("latitud", loc.getLatitud());
            hashMap.put("longitud", loc.getLongitud());
            hashMap.put("id_comarca", loc.getIdComarca());
        }
        return hashMap;
```

```
92      }
93
94      public PuntDePas getPuntDePasByIdUB(Integer id) {
95          PuntDePas tt = null;
96          for (Ruta r: tripUB.getRutes()) {
97              tt = r.getPuntDePasById(id);
98              if (tt!=null) return tt;
99          }
100         return tt;
101     }
102
103     public Localitat getLocalitatByIdUB(Integer id) {
104         Localitat tt = null;
105         for (Comarca c: tripUB.getAllComarques()) {
106             tt = c.getLocalitat(id);
107             if (tt!=null) return tt;
108         }
109         return tt;
110     }
111
112     public HashMap<Object, Object> getLocalitat_y_EtapaByID(Integer id) throws Exception {
113         Etapa etapa = getEtapaByIdUB(id);
114         Localitat localitat = getLocalitatByIdUB(etapa.getId());
115         // Por seguridad y comprension lo hacemos en 2 pasos, pero realmente, nuestra DB permite hacer Localitat localitat = dataService.getLoca
116         //al ser herencia recordar que id_localitat == id_etapa
117         HashMap<Object, Object> hashMap = new HashMap<>();
118         hashMap.put("id", etapa.getId());
119         hashMap.put("nom", localitat.getNom());
120         return hashMap;
121     }
122
123     public Etapa getEtapaByIdUB(Integer id) {
124         Etapa tt = null;
```

```
125         for (Ruta r: tripUB.getRutes()) {
126             tt = r.getEtapaById(id);
127             if (tt!=null) return tt;
128         }
129         return tt;
130     }
131 
132     public PuntDePas findPuntDePas(Integer idPuntDePas) {
133         PuntDePas pdp = null;
134         for (Ruta r: tripUB.getRutes()) {
135             pdp = r.findPuntDePas(idPuntDePas);
136             if (pdp != null) {
137                 return pdp;
138             }
139         }
140         return pdp;
141     }
142 
143     public Etapa findEtapa(Integer idEtapa) {
144         Etapa e = null;
145         for (Ruta r : tripUB.getRutes()) {
146             e = r.findEtapa(idEtapa);
147             if (e!= null) break;
148         }
149         return e;
150     }
151 
152 }
```

Listing 7: Classe *LocalitatsFacade* del nostre projecte.

```java
package ub.edu.model;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

public class PersonFacade {

    private TripUB tripUB;

    public static PersonFacade personFacade;

    private PersonFacade(TripUB tripUB) {
        this.tripUB = tripUB;
    }

    public static PersonFacade getInstance(TripUB tripUB) {
        if(personFacade == null){
            personFacade = new PersonFacade(tripUB);
        }
        return personFacade;
    }

    public StatusType validateRegistrePersona(String username, String password) {
        if (Seguretat.isMail((username)) && Seguretat.isPasswordSegur(password)) {
            Persona persona = findPersonaByCorreu(username);
            if (persona != null) {
                return StatusType.PERSONA_DUPLICADA; // Persona duplicada
            } else return StatusType.REGISTRE_VALID; // Registre valid
        } else return StatusType.FORMAT_INCORRECTE_CORREU_PWD; // Format incorrecte
    }

    public int loguejarPersona(String username, String password) {
```

```
34          Persona persona = findPersonaByCorreu(username);
35          if (persona == null) {
36              return 1; // "Correu inexistent";
37          }
38          if (persona.getPwd().equals(password)) {
39              return 0; // "Login correcte";
40          } else {
41              return 2; //"Contrassenya incorrecta";
42          }
43      }
44
45      public String recuperarContrasenya(String username) {
46          Persona persona = findPersonaByCorreu(username);
47          if (persona == null) {
48              return StatusType.CORREU_INEXISTENT.toString();
49          }
50          return persona.getPwd();
51      }
52
53      public StatusType loguejarSociStatus(String correu, String password) {
54          Persona persona = findPersonaByCorreu(correu);
55          if (persona == null) {
56              return StatusType.CORREU_INEXISTENT;
57          }
58          if (persona.getPwd().equals(password)) {
59              return StatusType.LOGIN_CORRECTE;
60          } else {
61              return StatusType.CONTRASENYA_INCORRECTA;
62          }
63      }
64
65      public List<HashMap<Object,Object>> getAllGrupsPerNom() {
66          List<HashMap<Object,Object>> grupsDisponibles = new ArrayList<>();
```

```
        //debido a que en el initGrups se inicializó y se guardo los grupos en this.grups
        //en lugar de volver a llamar a DB con el dataService.getAllGrups();
        //lo cogemos de la variable. De esta forma (si el proyecto fuera grande) ahorramos tiempo
        for (Grup g : tripUB.getAllGrups()) {
            HashMap<Object,Object> hashMap= new HashMap<>();
            hashMap.put("id",g.getId());
            hashMap.put("nom",g.getNom());
            grupsDisponibles.add(hashMap);
        }
        return grupsDisponibles;
    }

    public ArrayList<HashMap<Object,Object>> getAllGrupsPerPersona(String correu) {

        ArrayList<String> arrayNomGrupsByPersonaCorreu = tripUB.getXarxaPersones().getAllGrupsPerPersona(correu);
        ArrayList<HashMap<Object,Object>> listHashMap = new ArrayList<>();
        for (Grup g: tripUB.getAllGrups()) {
            if(arrayNomGrupsByPersonaCorreu.contains(g.getNom())){
                HashMap<Object,Object> hashMap = new HashMap<>();
                hashMap.put("id",g.getId());
                hashMap.put("nom",g.getNom());
                listHashMap.add(hashMap);
            }
        }
        return listHashMap;
    }

    public void addPersonaToGrup(Integer grup, String correuPersona) {
        Grup g = tripUB.findGrup(grup);
        Persona p = findPersonaByCorreu(correuPersona);
        PerfilPersona pp = p.getPerfil();
        pp.addGrup(g);
        g.addPersona(p);
```

```
100     }
101
102     public Persona findPersonaByCorreu(String correu) {
103         XarxaPersones xp = tripUB.getXarxaPersones();
104         return(xp.find(correu));
105     }
106
107     public boolean setXarxaPersones(List<Persona> l) {
108         XarxaPersones xarxa;
109         if (l != null) {
110             xarxa = new XarxaPersones(l);
111             tripUB.setXarxaPersones(xarxa);
112             return true;
113         } else return false;
114     }
115
116 }
```

Listing 8: Classe *PersonFacade* del nostre projecte.

```
1  package ub.edu.model;
2
3  import java.util.ArrayList;
4  import java.util.HashMap;
5  import java.util.List;
6
7  public class RutaFacade {
8
9      private TripUB tripUB;
10
11     public static RutaFacade rutaFacade;
12
```

```
13      private RutaFacade(TripUB tripUB) {
14          this.tripUB = tripUB;
15      }
16
17      public static RutaFacade getInstance(TripUB tripUB) {
18          if(rutaFacade == null){
19              rutaFacade = new RutaFacade(tripUB);
20          }
21          return rutaFacade;
22      }
23
24      public List<HashMap<Object,Object>> llistarRutesPerLocalitat(String nomLocalitat) {
25          ArrayList<HashMap<Object,Object>> lRuta = new ArrayList<>();
26          HashMap<Object, Object> atributRuta;
27
28          for (Ruta ruta : tripUB.getRutes()) {
29              if(ruta.getLocalitatOrigen().getNom().equals(nomLocalitat) || ruta.getLocalitatDesti().getNom().equals(nomLocalitat)) {
30                  atributRuta = new HashMap<>();
31                  Integer id = ruta.getId();
32                  String nom = ruta.getNom();
33                  atributRuta.put("id", id);
34                  atributRuta.put("nom", nom);
35                  lRuta.add(atributRuta);
36              }
37          }
38          return lRuta;
39      }
40
41      public List<HashMap<Object,Object>> llistarRutesPerComarca(String nomComarca) {
42          ArrayList<HashMap<Object,Object>> lRuta = new ArrayList<>();
43          HashMap<Object, Object> atributRuta;
44
45          for (Ruta ruta : tripUB.getRutes()) {
```

```
            for(Comarca comarca: ruta.getComarques()){
                if(comarca.getNom().equals(nomComarca)){
                    atributRuta = new HashMap<>();
                    Integer id = ruta.getId();
                    String nom = ruta.getNom();
                    atributRuta.put("id", id);
                    atributRuta.put("nom", nom);
                    lRuta.add(atributRuta);
                }
            }
        }
        return lRuta;
    }

    public List<HashMap<Object,Object>> getAllRutesPerNom() {
        List<HashMap<Object, Object>> rutesDisponibles = new ArrayList<>();
        for (Ruta r : tripUB.getRutes()) {
            HashMap<Object, Object> atributRuta = new HashMap<>();
            Integer id = r.getId();
            String nom = r.getNom();
            atributRuta.put("id", id);
            atributRuta.put("nom", nom);

            rutesDisponibles.add(atributRuta);
        }
        return rutesDisponibles;
    }

    public HashMap<Object, Object> getRutaById(Integer id_ruta) throws Exception {
        Ruta r = tripUB.findRuta(id_ruta);
        HashMap<Object, Object> hashMap = new HashMap<>();
        if (r != null) {
            hashMap.put("id", r.getId());
```

```
            hashMap.put("nom", r.getNom());
            hashMap.put("dataCreacio", r.getDataCreacio());
            hashMap.put("durada", r.getDurada());
            hashMap.put("descripcio", r.getDescripcio());
            hashMap.put("cost", r.getCost());
            hashMap.put("dificultat", r.getDificultat());
            hashMap.put("tipusRuta", r.getTipusRuta());
            hashMap.put("id_lloc_origen", r.getIdLlocOrigen());
            hashMap.put("id_lloc_desti", r.getIdLlocDesti());
        }
        return hashMap;
    }


}
```

Listing 9: Classe *RutaFacade* del nostre projecte.

```
package ub.edu.model;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Objects;

public class TramFacade {
    private TripUB tripUB;

    public static TramFacade tramFacade;

    private TramFacade(TripUB tripUB) {
        this.tripUB = tripUB;
```

```
    }

    public static TramFacade getInstance(TripUB tripUB) {
        if(tramFacade == null){
            tramFacade = new TramFacade(tripUB);
        }
        return tramFacade;
    }

    public List<HashMap<Object, Object>> getAllTramsEtapaTrackByRutaId(Integer id_ruta) throws Exception {
        //Obtener todos los trams -> si se cogen directamente los TramEtapa y TtramTrack, dentro ya estarán los Trams respectivamente

        List<HashMap<Object, Object>> listaFinal = new ArrayList<>();

        // al ser una herencia, y tal como está montada la DB
        // dentro de la hija, la madre contiene tambien los datos respectivos a la madre/super
        // el id de la madre, es igual al id de la hija
        // TramEtapa y TramTrack son hijas de Tram
        // por tanto, tendrán el mismo id

        for (TramEtapa tramEtapa : getAllTramEtapasUB()) {
            HashMap<Object, Object> hashMap = new HashMap<>();
            if (Objects.equals(tramEtapa.getId_ruta(), id_ruta)) {
                String s = "Tram Etapa " + tramEtapa.getId().toString();
                hashMap.put("id", tramEtapa.getId());
                hashMap.put("value", 1); //el value es un flag que indica si es TramEtapa o TramTrack
                hashMap.put("text", s);
                listaFinal.add(hashMap);
            }
        }
        for (TramTrack tramTrack : getAllTramTrackUB()) {
            HashMap<Object, Object> hashMap = new HashMap<>();
            if (Objects.equals(tramTrack.getId_ruta(), id_ruta)) {
```

```
                String s = "Tram Track " + tramTrack.getId().toString();
                hashMap.put("id", tramTrack.getId());
                hashMap.put("value", 2); //el value es un flag que indica si es TramEtapa o TramTrack
                hashMap.put("text", s);
                listaFinal.add(hashMap);
            }
        }
        return listaFinal;
    }

    public HashMap<Object, Object> getTramTrackById(Integer id) throws Exception {
        TramTrack tramTrack = getTramTrackByIdUB(id);
        HashMap<Object, Object> hashMap = new HashMap<>();
        hashMap.put("id", tramTrack.getId());
        hashMap.put("id_pdp_origen", tramTrack.getIdPuntDePasOrigen());
        hashMap.put("id_pdp_desti", tramTrack.getIdPuntDePasDesti());
        return hashMap;
    }

    public HashMap<Object, Object> getTramEtapaById(Integer id) throws Exception {
        TramEtapa tramEtapa = getTramEtapaByIdUB(id);
        HashMap<Object, Object> hashMap = new HashMap<>();
        hashMap.put("id", tramEtapa.getId());
        hashMap.put("id_etapa_origen", tramEtapa.getIdEtapaOrigen());
        hashMap.put("id_etapa_desti", tramEtapa.getIdEtapaDesti());
        return hashMap;
    }

    private Localitat getLocalitatByIdUB(Integer id) {
        Localitat tt = null;
        for (Comarca c: tripUB.getAllComarques()) {
            tt = c.getLocalitat(id);
            if (tt!=null) return tt;
```

```
81          }
82          return tt;
83      }
84
85      private Etapa getEtapaByIdUB(Integer id) {
86          Etapa tt = null;
87          for (Ruta r: tripUB.getRutes()) {
88              tt = r.getEtapaById(id);
89              if (tt!=null) return tt;
90          }
91          return tt;
92      }
93
94      public HashMap<Object, Object> getLocalitat_y_EtapaByID(Integer id) throws Exception {
95          Etapa etapa = getEtapaByIdUB(id);
96          Localitat localitat = getLocalitatByIdUB(etapa.getId());
97          // Por seguridad y comprension lo hacemos en 2 pasos, pero realmente, nuestra DB permite hacer Localitat localitat = dataService.getLoca
98          //al ser herencia recordar que id_localitat == id_etapa
99          HashMap<Object, Object> hashMap = new HashMap<>();
100         hashMap.put("id", etapa.getId());
101         hashMap.put("nom", localitat.getNom());
102         return hashMap;
103     }
104
105     private List<TramEtapa> getAllTramEtapa() throws Exception {
106         return getAllTramEtapasUB();
107     }
108
109     private List<TramTrack> getAllTramTrack() throws Exception {
110         return getAllTramTrackUB();
111     }
112
113     public List<TramEtapa> getAllTramEtapasUB() {
```

```
114         List<TramEtapa> listTrams = new ArrayList<>();
115         for (Ruta r: tripUB.getRutes()) {
116             List<TramEtapa> le = r.getAllTramEtapas();
117             for (TramEtapa te: le) {
118                 listTrams.add(te);
119             }
120         }
121         return listTrams;
122     }
123     public List<TramTrack> getAllTramTrackUB() {
124         List<TramTrack> listTrams = new ArrayList<>();
125         for (Ruta r: tripUB.getRutes()) {
126             List<TramTrack> le = r.getAllTramTracks();
127             for (TramTrack te: le) {
128                 listTrams.add(te);
129             }
130         }
131         return listTrams;
132     }
133 
134     public TramTrack getTramTrackByIdUB(Integer id) {
135         TramTrack tt = null;
136         for (Ruta r: tripUB.getRutes()) {
137             tt = r.findTramTrackById(id);
138             if (tt!=null) return tt;
139         }
140         return tt;
141     }
142 
143     public TramEtapa getTramEtapaByIdUB(Integer id) {
144         TramEtapa tt = null;
145         for (Ruta r: tripUB.getRutes()) {
146             tt = r.findTramEtapaById(id);
```

```
147             if (tt!=null) return tt;
148         }
149         return tt;
150     }
151 
152     public void addTram(Tram te) {
153         Ruta r = tripUB.findRuta(te.getId_ruta());
154         if (r!=null)
155             r.addTram(te);
156     }
157 
158     public void addTransportTram(Integer id_tram, Transport t) {
159 
160         for (Ruta r: tripUB.getRutes()) {
161             if (r.findTram(id_tram)) {
162                 r.addTransportTram(id_tram, t);
163                 break;
164             }
165         }
166     }
167 
168 }
```

Listing 10: Classe *TramFacade* del nostre projecte.

```
1 package ub.edu.model;
2 
3 import ub.edu.resources.dao.Parell;
4 
5 import java.util.*;
6 import java.util.function.Function;
7 import java.util.stream.Collectors;
```

```
8
9   public class TripUB {
10      private Integer id;
11      private XarxaPersones xarxaPersones;
12      private Map<Integer, Ruta> rutaMap;
13      private Map<Integer, Comarca> comarcaMap;
14      private Map<Integer, Grup> grups;
15      private Map<Integer, Allotjament> allotjamentsMap;
16
17      private Map<Integer, Reserva> reservaMap;
18
19      private static TripUB tripUB;
20
21      private TripUB() {
22          grups = new HashMap<>();
23          comarcaMap = new HashMap<>();
24          rutaMap = new HashMap<>();
25          allotjamentsMap = new HashMap<>();
26          reservaMap = new HashMap<>();
27      }
28
29      public static TripUB getInstance(){
30          if(tripUB == null){
31              tripUB = new TripUB();
32          }
33          return tripUB;
34      }
35
36
37      public boolean setGrups(List<Grup> lg) {
38          this.grups =  (lg.stream()
39                  .collect(Collectors.toMap(Grup::getId, Function.identity())));
40          return (grups!= null);
```

```
41      }
42
43      public void setXarxaPersones(XarxaPersones x) {
44          this.xarxaPersones = x;
45      }
46
47      public boolean setRutes(List<Ruta> listRutas) {
48          this.rutaMap =  (listRutas.stream()
49                  .collect(Collectors.toMap(Ruta::getId, Function.identity())));
50          return (this.rutaMap != null);
51      }
52
53      public boolean setAllotjaments(List<Allotjament> listAllotjaments){
54          this.allotjamentsMap = (listAllotjaments.stream()
55                  .collect(Collectors.toMap(Allotjament::getId, Function.identity())));
56          return (this.allotjamentsMap != null);
57      }
58
59      public boolean setReserves(List<Reserva> reservaList){
60          this.reservaMap = (reservaList.stream()
61                  .collect(Collectors.toMap(Reserva::getId, Function.identity())));
62          return (this.reservaMap != null);
63      }
64
65      public boolean  setComarques(List<Comarca> listComarques) {
66          comarcaMap =  (listComarques.stream()
67                  .collect(Collectors.toMap(Comarca::getId, Function.identity())));
68          return (comarcaMap != null) ;
69      }
70      public Integer getId() {
71          return id;
72      }
73
```

```
74      public void setComarquesToRutes(List<Parell<Integer, Integer>> relacionsRC) {
75          for (Parell p : relacionsRC) {
76              Ruta r = rutaMap.get(p.getElement2());
77              Comarca c = comarcaMap.get(p.getElement1());
78              r.addComarca(c);
79          }
80      }
81      public void setId(Integer id) {
82          this.id = id;
83      }
84
85      public Comarca findComarca(Integer idComarca) {
86          Comarca c = null;
87          c = comarcaMap.get(idComarca);
88          return c;
89      }
90
91      public Grup findGrup(Integer idGrup) {
92          return grups.get(idGrup);
93      }
94
95      public Ruta findRuta(Integer idRuta) {
96          return rutaMap.get(idRuta);
97      }
98
99      public List<Comarca> getAllComarques() {
100         //no se puede hacer cast directo entre map->hasmap a una collection->list. Se debe instanciar una nueva arraylist
101         return new ArrayList<Comarca>(this.comarcaMap.values());
102     }
103
104     public void addGrup(Grup grup) {
105         grups.put(grup.getId(), grup);
106     }
```

```
107
108
109     public List<Ruta> getRutes() {
110         //no se puede hacer cast directo entre map->hasmap a una collection->list. Se debe instanciar una nueva arraylist
111         return new ArrayList<Ruta>(this.rutaMap.values());
112     }
113
114     public XarxaPersones getXarxaPersones() {
115         return xarxaPersones;
116     }
117
118     public List<Grup> getAllGrups() {
119         //no se puede hacer cast directo entre map->hasmap a una collection->list. Se debe instanciar una nueva arraylist
120         return new ArrayList<Grup>(this.grups.values());
121     }
122
123     public Integer getIdGrupByName(String newValue) {
124         List<Grup> list = this.getAllGrups();
125         for (Grup g: list) {
126             if(g.getNom().equals(newValue))
127                 return g.getId();
128         }
129         return null;
130     }
131
132     public List<Ruta> findRutaByOrigen(Integer id) {
133         List<Ruta> rutes = new ArrayList<>();
134         for (Ruta r: rutaMap.values()) {
135             if (r.getIdLlocOrigen()==id) rutes.add(r);
136         }
137         return rutes;
138     }
139
```

```
140     public List<Ruta> findRutaByDesti(Integer id) {
141         List<Ruta> rutes = new ArrayList<>();
142         for (Ruta r: rutaMap.values()) {
143             if (r.getIdLlocDesti()==id) rutes.add(r);
144         }
145         return rutes;
146     }
147 
148     public Allotjament getAllotjament(Integer idAllotjament){
149         return allotjamentsMap.get(idAllotjament);
150     }
151 
152     public Ruta getRuta(Integer idRuta){
153         return rutaMap.get(idRuta);
154     }
155 
156     public Persona findPersona(String correuPersona) {
157         return xarxaPersones.find(correuPersona);
158     }
159 
160     public Collection<Allotjament> getAllAllotjaments() { return allotjamentsMap.values(); }
161 
162     public List<Tram> getAllTrams() {
163         HashMap<Integer,Tram> tramList = new HashMap<>();
164         for(Ruta ruta : this.rutaMap.values()) {
165             // elimina automàticament aquells trams amb ids duplicats (els que són iguals, vaja) //
166             tramList.putAll(ruta.getAllTrams());
167         }
168         return new ArrayList<>(tramList.values());
169     }
170 
171     public List<Reserva> getReservesByAllotjament(Integer idAllotjament){
172         List<Reserva> llista = new ArrayList<>();
```

```
173
174         for(Reserva r: reservaMap.values()){
175             if(r.getIdAllotjament().equals(idAllotjament)){
176                 llista.add(r);
177             }
178         }
179         return llista;
180     }
181
182     public void addReservaToPersona(String nomPersona, Reserva reserva) throws Exception {
183         if(reserva == null) throw new Exception("No es pot afegir una reserva nul·la");
184         Persona persona = this.xarxaPersones.find(nomPersona);
185         if(persona == null) throw new Exception("Aquesta persona no existeix");
186         persona.addReserva(reserva);
187     }
188
189     public void addReserva(Reserva reserva) throws Exception {
190         if(reserva == null) throw new Exception("No es pot afegir una reserva nul·la");
191         reservaMap.put(reserva.getId(),reserva);
192     }
193
194     public void removeReserva(Reserva reserva) throws Exception {
195         if(reserva == null) throw new Exception("No es pot anul·lar una reserva nul·la");
196         if(reservaMap.get(reserva.getId()) == null) throw new Exception("Aquesta reserva no es troba a la llista de reserves");
197         reservaMap.remove(reserva.getId());
198     }
199
200     public void removeReservaFromPersona(String nomPersona, Reserva reserva) throws Exception {
201         if(reserva == null) throw new Exception("No es pot afegir una reserva nul·la");
202         Persona persona = this.xarxaPersones.find(nomPersona);
203         if(persona == null) throw new Exception("Aquesta persona no existeix");
204         persona.removeReserva(reserva);
205     }
```

```
206
207 }
208
```

Listing 11: Classe *TripUB* del nostre projecte.

2.2

Millores

Les millores que hem fet respecte la pràctica 3, en quant a codi, són poques. Hem aplicat nous patrons que veurem més endavant en aquesta memòria. El codi base que se'ns ha proporcionat és molt pobre en quant a bones pràctiques de codi: hi ha molt d'acoblament, està ple de codi espagueti i es vulneren altres principis molt importants. *Com hem comentat amb el nostre professor de laboratori, solament és objecte d'avaluació les classes que interactuen directament en la pràctica actual, i la resta no les hem tocat*.

La millora principal, segurament, sigui la interacció amb una vista mitjançant el patró MVC. Considerem que el nostre codi de la pràctica 3 complia, en la mesura del possible, tots els principis de disseny de software que hem après al llarg del curs.

GRASP (General Responsibility Assignment Software Patterns) és un conjunt de patrons de disseny que ajuden a assignar responsabilitats correctament a les classes d'un sistema de manera que es puguin mantenir i evolucionar fàcilment. Aquests patrons inclouen:

- **Creator**: assigna la responsabilitat de crear objectes a la classe que té la informació necessària per fer-ho.
- **Controller**: assigna la responsabilitat de gestionar la interacció amb un usuari o un sistema extern a una classe.
- **Low Coupling**: minimitza les dependències entre classes per reduir la complexitat i millorar la mantenibilitat.
- **High Cohesion**: maximitza la cohesió dins de les classes per millorar la seva comprensió i reutilització.
- **Indirection**: utilitza una capa intermediària per a la comunicació entre classes per evitar dependències directes.

SOLID és un conjunt de principis de disseny que ajuden a construir sistemes flexibles i mantenibles. Aquests principis inclouen:

- **Single Responsibility Principle**: cada classe ha de tenir una única responsabilitat i totes les seves funcions han de ser relacionades amb aquesta responsabilitat.
- **Open-Closed Principle**: les classes han de ser obertes per a l'extensió, però tancades per a la modificació.
- **Liskov Substitution Principle**: les subclasses han de poder substituir les seves superclasses sense alterar el comportament del sistema.
- **Interface Segregation Principle**: les interfaces han de ser específiques i no haurien de contenir mètodes innecessaris per a algunes de les seves implementacions.

- **Dependency Inversion Principle**: les classes no han de dependre de detalls de implementació, sinó de les interfaces abstractes.

També hem aplicat altres patrons de disseny, com:

1. El patró de disseny *Abstract Factory*, que proporciona una interfície per a la creació de famílies d'objectes relacionats o depenents sense especificar les seves classes concretes. Això permet canviar fàcilment el conjunt d'objectes que es creen segons les necessitats del sistema. *L'hem usat a la capa de persistència i al pagament*.
2. El patró de disseny *Strategy* defineix una família d'algorismes, encapsula cadascun d'ells i els fa intercanviables. Això permet que l'algorisme utilitzat pugui ser seleccionat en temps d'execució i que es puguin afegir noves opcions d'algorismes fàcilment sense alterar la resta del codi. Aquest patró és útil quan hi ha diverses opcions per resoldre un problema i es vol poder canviar entre elles fàcilment.
3. El patró de disseny *Singleton*, que garanteix que només hi ha una instància d'una classe en tot el programa i proporciona un punt d'accés global a aquesta instància. Això és útil quan es vol compartir dades o estat entre diverses parts del programa o quan es vol limitar l'ús d'un recurs compartit. *L'hem usat a moltes classes; entre d'elles, el Controlador i les Facades, així com a la capa de persistència*.
4. El patró de disseny *Composite* és un patró estructural que permet construir arbres d'objectes on tots els nodes de l'arbre tenen una interfície comuna. Això permet tractar els nodes individualment o en conjunt de manera transparent. Aquest patró és útil per a representar jerarquies d'objectes amb comportaments similars.
5. El patró de disseny *Facade* és un patró de comportament que proporciona una interfície simplificada per a un conjunt de classes, amagant la complexitat del sistema. Això permet que les classes client puguin utilitzar les funcionalitats del sistema de manera més senzilla i evitar dependències directes amb les classes del sistema. Aquest patró és útil per a millorar l'ús i la comprensió del sistema per part dels clients. *En les nostres pràctiques anteriors, l'hem vist en les Façanes del Model:* `AllotjamentFacana`, `LlistesFacana`, `CercadorFacana`, `LoginFacana`, `RegisterFacana`, `RutaFacana`, `TramFacana` i `TransportFacana`. En la pràctica actual també en tenim, però diferents.

2.3 Nous patrons

El patró Observer estableix una relació unidireccional en la qual un objecte, anomenat subjecte, notifica als altres objectes, anomenats observadors, quan es produeixen canvis en ell. Això permet que els observadors

puguin reaccionar de manera apropiat a aquests canvis sense tenir una dependència directa amb el subjecte. Sovint s'utilitza per implementar sistemes de gestió d'esdeveniments distribuïts, en programari "basat en esdeveniments".

L'aplicació del patró a les classes de *Teoria* no té res a veure amb com l'hem hagut d'aplicar a la pràctica actual. En el context de Java i utilitzant el patró observador entre escenes, es podria implementar fent servir les classes `java.util.Observer` i `java.util.Observable`. Com que les escenes ja hereten d'una classe `Escena`, i `Java` no permet l'herència múltiple, no podem usar-les. En el codi base, les obtenim a partir d'una factoria, però el funcionament és similar que el que veurem ara. En aquest exemple, `Scene2` és un observador de `Scene1`. Quan l'estat de `Scene1` canvia, crida el mètode `setChanged()` per indicar que ha canviat d'estat, i després crida el mètode `notifyObservers()` per notificar els seus observadors del canvi. `Scene2` implementa el mètode `update()`, que és cridat per `Scene1` quan canvia d'estat. En el mètode update(), Scene2 pot actualitzar el seu propi estat en funció de l'estat de `Scene1`. Hem hagut d'aplicar un esquema semblant al següent:

```
public interface SceneObserver {
  void setSubject(Scene1 subject);
  void update();
}

public class Scene1 {
  private List<SceneObserver> observers = new ArrayList<>();
  private String state;

  public void addObserver(SceneObserver observer) {
    observers.add(observer);
    observer.setSubject(this);
  }

  public void removeObserver(SceneObserver observer) {
    observers.remove(observer);
  }

  public void setState(String state) {
    this.state = state;
    notifyObservers();
  }

  private void notifyObservers() {
    for (SceneObserver observer : observers) {
      observer.update();
```

```
    }
  }
}

public class Scene2 implements SceneObserver {
  private Scene1 scene1;

  public void setSubject(Scene1 subject) {
    this.scene1 = subject;
  }

  public void update() {
    String state = scene1.getState();
    // Update the state of Scene2 based on the state of Scene1
  }
}
public class Scene2 implements SceneObserver {
  private Scene1 scene1;

  public void setSubject(Scene1 subject) {
    this.scene1 = subject;
  }

  public void update() {
    String state = scene1.getState();
    // Update the state of Scene2 based on the state of Scene1
  }
}
```

Com que `EscenaMain` vol observar el que passa a `EscenaReservarAllotjament`, seria ideal que `Scene1` fes el paper de la segona i `Scene2` el de la primera. La realitat és més complicada, ja que, com hem dit, no s'instancien aquestes classes de manera habitual, sinó que s'obtenen a partir de factories. Per tant, no podem dependre de constructors. Una dificultat afegida és que és la pròpia escena `EscenaMain` la que va creant les altres escenes, i `EscenaReservarAllotjament` ve creada per una escena que crea `EscenaMain`. En altres paraules, `EscenaMain` no influeix de manera directa a `EscenaReservarAllotjament`.

Nosaltres hem argumentat que `EscenaMain` observa totes aquestes escenes intermitges, i hem pogut lincar amb una relació d'observació aquestes dues. *Es pot veure al nostre codi com ho hem fet*. Hem fet una interfície pels mètodes observadors (`SceneObserver`) i un altre pels observables (`SceneObservable`).

Per últim, aquest patró afecta a la història d'usuari de reservar i cancel·lar un allotjament i a la visualització del *top-5* allotjaments més reservats. En efecte, al fer un canvi en les reserves és probable que el *top* es vegi modificat. Si ens posem puristes, com fem d'observadors sobre un munt d'escenes intermitges es podrien afegir les històries de llistar rutes i visualitzar detalls (tant de rutes com de trams, punts de pas...).

2.4 Model de Domini

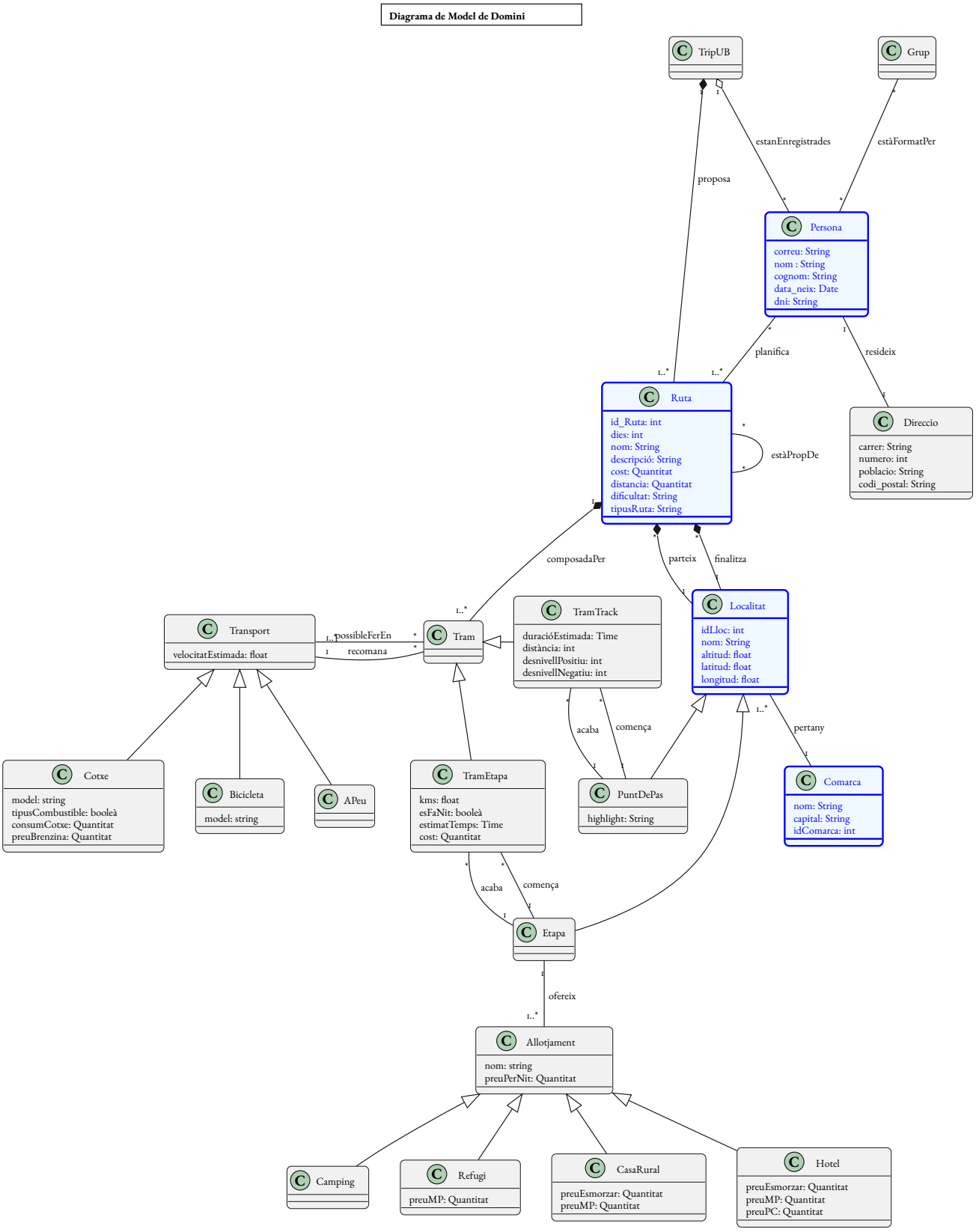


Figura 2.1: Diagrama del nostre model de domini resultant, de la pràctica 1.

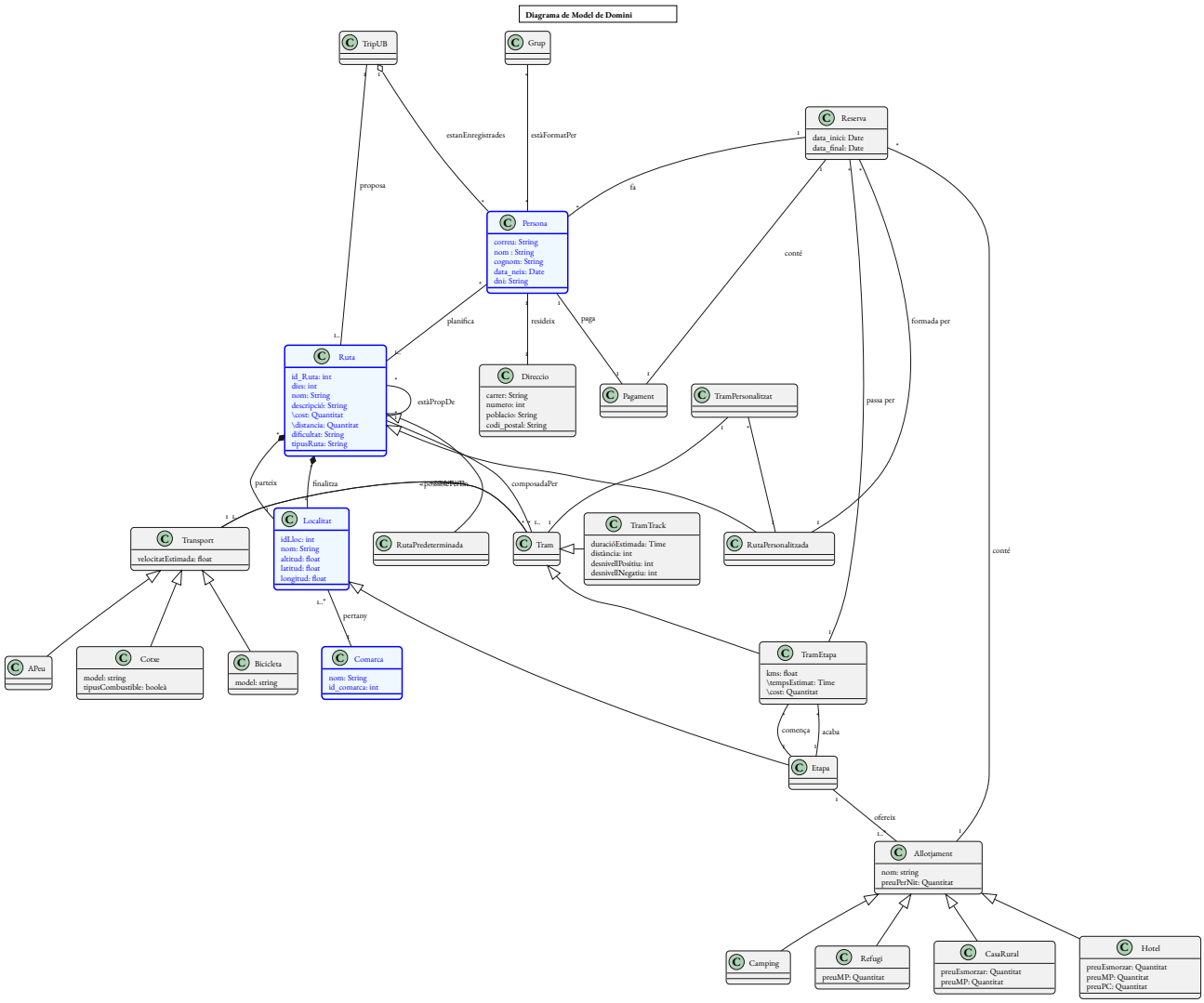


Figura 2.2: Diagrama del nostre model de domini resultant, de la pràctica 3.

Es pot ampliar el diagrama tant com es vulgui. Està fet a LaTeX i es veu a resolució completa. El nostre model de domini ha canviat respecte la primera pràctica. *Donat que el codi de la tercera pràctica i el de la quarta són tan diferents, i que el volum de codi que queda fora de l'abast d'aquesta última és molt gran, triarem analitzar el de la tercera pràctica.*

Segons les correccions que hem anat rebent, havíem de distingir entre `RutaPredeterminada` i `RutaPersonalitzada` dins del context d'una `Ruta`. El mateix passa amb `TramPersonalitzat`, tot i que aquesta vegada no parlem de `TramPredeterminat`. També vam treure `PuntdePas` perquè no enteníem ben bé la seva funcionalitat. Hem afegit les classes `Pagament` i `Reserva`.

En definitiva, l'aspecte extern del model de domini ha canviat, no molt; però és important remarcar que, internament, les classes difereixen molt: el codi s'ha anat refinant. D'això no en parlarem aquí, ja que no correspon a cap requeriment del model de domini.

2.5

Diagrames de classe

El diagrama de classes en PlantUML és una representació gràfica que es fa servir en el disseny d'un projecte de programari per a especificar les classes, atributs i mètodes que formen part del sistema. Un diagrama de classes es compon de diferents elements, com ara:

1. **Classes**: són les unitats bàsiques del sistema que representen les diferents entitats del negoci. Cada classe té un nom, atributs i mètodes.
2. **Atributs**: són les variables que conformen una classe i que descriuen les seves propietats o característiques.
3. **Mètodes**: són les funcions que es poden cridar a les classes i que s'encarreguen de realitzar tasques específiques.

Un diagrama de classes es pot fer servir per a:

1. Especificar les relacions entre les diferents classes del sistema.
2. Identificar els atributs i mètodes de cada classe.
3. Defineix les responsabilitats de cada classe i les relacions que tenen amb les altres.
4. Facilita la comprensió del sistema a nivell global i permet identificar possibles problemes o errors en el disseny.

En resum, el diagrama de classes és una eina molt útil en el desenvolupament d'un projecte de programari ja que ens permet visualitzar de manera clara i senzilla la estructura del sistema i les relacions entre les diferents classes. Això ens ajuda a identificar problemes en el disseny i a planificar millor el desenvolupament del projecte.

Comentarem els diagrames de classe i també els deixarem a la carpeta `doc` del projecte. Distingirem entre els tres paquets que tenim: `resources`, `model` i `controller`. En aquesta memòria ens centrarem en els dos últims, ja que són els que més hem editat.

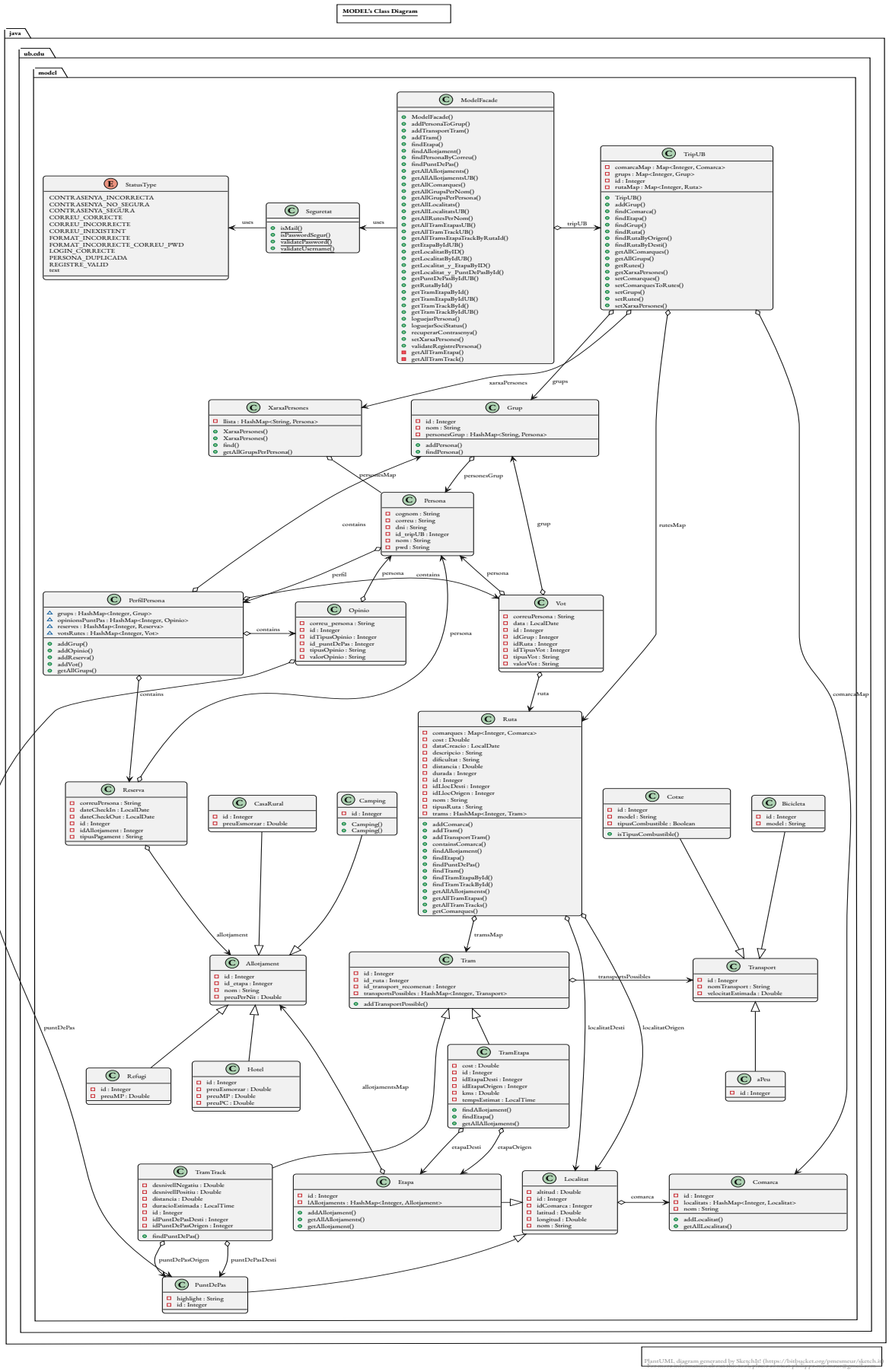


Figura 2.3: Diagrama del nostre diagrama de classes al *model*, abans de començar la quarta pràctica.

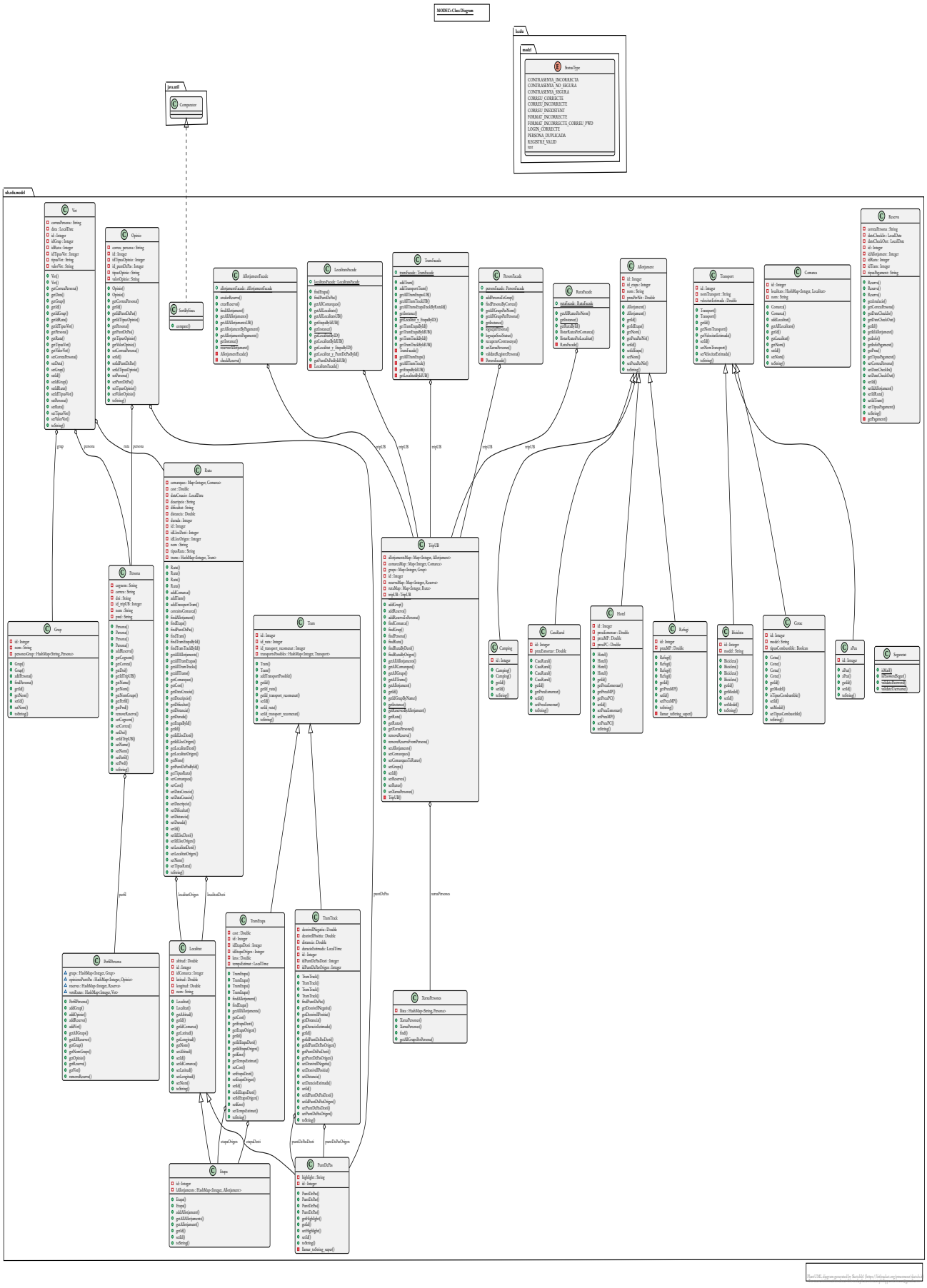


Figura 2.4: Diagrama de classes al *model*, després d'acabar la quarta pràctica.

Les diferències són clares. Hem aplicat els patrons de disseny adients **a les classes que utilitzem per a les nostres funcionalitats**. Ja hem comentat que els patrons `AbstractFactory` i `Facade` ja venien aplicats a la capa de persistència i al model, respectivament.

A `Reserva` i `Allotjament`, entre d'altres, se'ls ha aplicat el patró *Baix Acoblament i Alta Cohesió*: utilitzaven un munt d'informació que no calia i que es podia obtenir per altres mitjans. Ens hem plantejat dividir `ModelFacade` en diverses façanes, i ho hem decidit canviar per dues raons: (1) se'ns ha deixat carta blanca per canviar tot el que veiéssim convenient, considerem que aquesta façana del model és fonamental per al seu funcionament i hem decidit dividir-la per no vulnerar el Single Responsability i la Classe Déu, i (2) més façanes implica més atributs de tipus `Facade` en certes classes i, per tant, augmentem lleugerament acoblament, però guanyem en flexibilitat del codi. Notem que hem intentat minimitzar l'ús de relacions, perquè eren molt farragoses de crear i en moltes circumstàncies no eren ni necessàries.

El diagrama de classes sencer, `p4-viewdb-tripub-c01.plantuml`, es troba directament al nostre projecte. No l'adjuntarem aquí perquè el fitxer té una resolució poc compatible amb aquest document. **De la mateixa manera, es poden trobar tots els diagrames que s'han anat veient al codi del nostre projecte**.

III

Conclusions

En aquesta pràctica hem tingut l'oportunitat de reforçar els nostres coneixements sobre la programació orientada a testos, una tècnica important per a garantir la qualitat dels nostres programes informàtics. Hem après a utilitzar els models de domini per a representar els conceptes clau del nostre sistema i a escriure històries d'usuari per a descriure els requisits dels nostres clients. També hem après a dissenyar casos d'ús per a especificar els diferents camins que poden seguir els nostres usuaris en l'ús del nostre sistema.

Hem treballat amb les eines necessàries per a implementar aquest tipus d'aplicacions, com ara llenguatges de programació orientats a objectes, frameworks d'interfície gràfica i eines de gestió de bases de dades. Hem après a aplicar el patró Model-Vista-Controlador per a dividir la nostra aplicació en tres capes distintes, cada una amb una responsabilitat específica, i a utilitzar les eines adequades per a implementar cada capa.

Hem tingut, per últim, alguns dubtes de com orientar els nostre codi, però sembla que al final ens n'hem sortit bé. Per altra banda, si hem de destacar una cosa que hem fet bé és intentar adaptar-nos a les bones pràctiques de programació des del principi: hem usat construccions genèriques sempre que hem pogut, hem adaptat els noms de les variables a les convencions de `Java` i hem intentat estalviar línies de codi innecessàries, així com fer un control d'errors robust i descentralitzat en les diferents classes, per fer-ho més heurístic.

En definitiva, hem assolit l'objectiu del programa que se'ns proposava a l'inici de la pràctica.

El repartiment de feina entre membres del grup s'ha fet de manera equitativa i barrejada, no podem delimitar ben bé què ha fet cadascú perquè una gran part l'hem feta junts.

Bibliografia

[Bec03] Kent Beck. *Test-driven development : by example*. eng. The Addison-Wesley signature series. Boston [etc: Addison-Wesley, 2003. isbn: 0321146530.

[Lew07] John Lewis. *Java software solutions : foundations of program design*. eng. 5th ed. Boston: Pearson Addison-Wesley, 2007. isbn: 9780321409492.

[Lar16] Craig Larman. *Applying UML and patterns : an introduction to object-oriented analysis and design and iterative development*. eng. Third edition. Chennai: Pearson, 2016. isbn: 9789332553941.