

PROJECTE MP3UF5

Llibreries de classes fonamentals en java



Elies Fatsini Camacho

2n DAM

Institut de l'Ebre

INDEX

Fase 1.....	3
Explicació del Projecte.....	3
Fase1.....	3
Fase 2.....	5
FASE 3.....	29
FASE 4.....	32
FASE 5.....	34

Fase 1

Explicació del Projecte

Aquest projecte anirà sobre l'estructura d'una competició de vehicles, com per exemple la F1.

L'estructura del projecte en aquestes primeres fases serà un simple programa en GUI utilitzant l'estructura MODEL/VISTA/CONTROLADOR, que tingui un formulari on introduir dades, modificar-les, eliminar-les i inclús filtrar per els diferents camps que tingui.

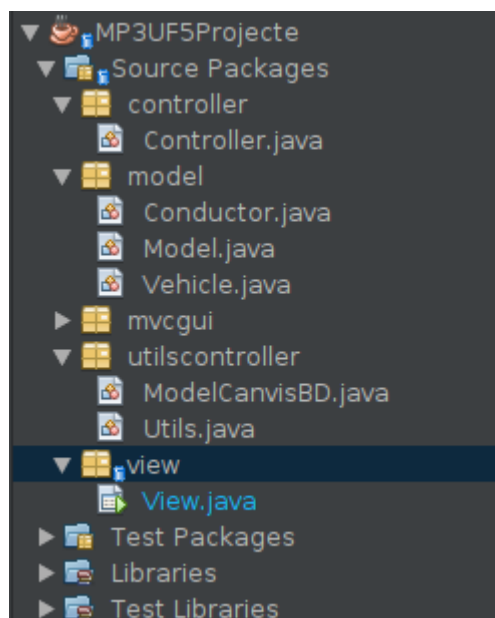
Al haver començat directament fent la fase 2 (degut a un retràs que vaig tenir amb el netbeans, que no hem va permetre arribar a l'entrega de la fase 1 bè), hi haurà una falta de documentació de la fase 1, tant en captures com en explicació. Així que la fase 2 serà una fusió de fase 1 i 2, on explicare els bàsics (com com insertar dades a la taula) i els específics de la fase 2(us de col·leccions).Disculpes.

Fase1

Començarem creant el projecte.

Tindrem diferents paquets principals: controller (Controlador), model (Model), mvcgui i view (Vista). Es distribuiran de la següent forma:

- controller: Tindrà el fitxer principal del controlador (Controller.java)
- model: Aquí es troben els diferents models de els diferents objectes que creem, apart del fitxer principal del Model.
- Mvogui: Només tindrà un fitxer MVCGUI.java, que servirà per a a juntar el controlador amb el model i la vista.
- View: Aquí es on anirà la vista. Els elements de la GUI i els metodes per a accedir a ells.



Vista principal

Aquest serà el disseny principal de la primera fase. Una taula on es mostrarà informació d'alguns vehicles predeterminats, juntament amb els vehicles que introdueixqui l'usuari amb el formulari que hi ha localitzat a la part esquerra, baix de la taula. El que hi ha al costat ens servirà per a editar registres ja entrats.

I els elements de la dreta ens mostraràn el títol de la taula, un botó per a eliminar una fila, un desplegable per a filtrar per diferents camps i el botó d'editar.

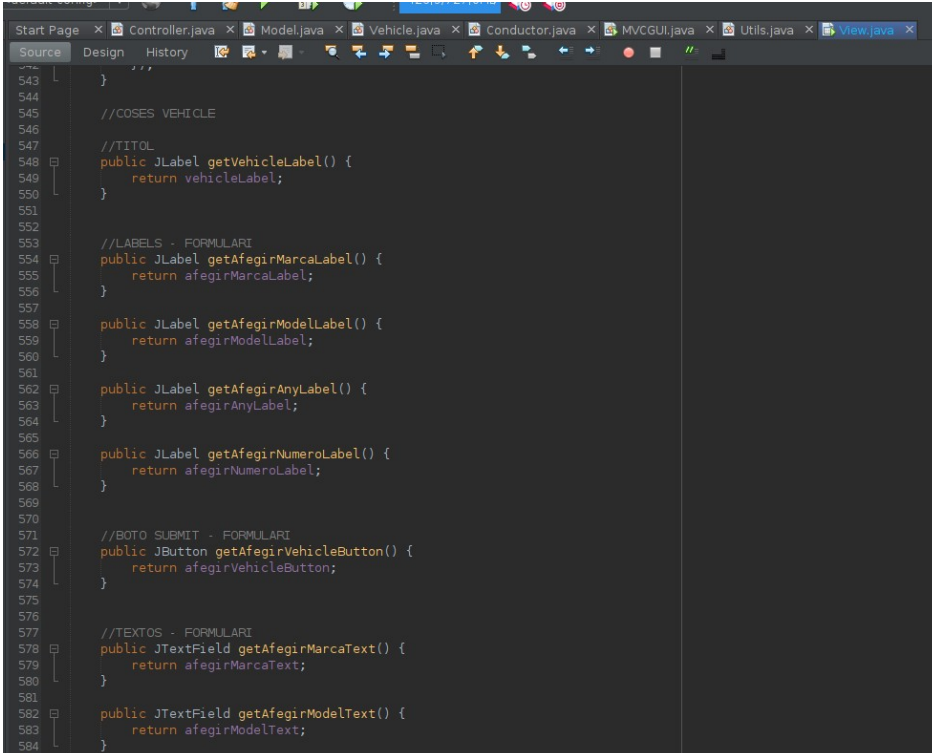
The mockup illustrates the main view layout. It features a central table with four columns labeled 'Title 1', 'Title 2', 'Title 3', and 'Title 4'. To the left of the table is a sidebar containing a form with labels (jLabel1, jLabel2, jLabel3, jLabel4) and text fields (jTextField1, jTextField2, jTextField3, jTextField4), along with a button (jButton1). To the right of the table is another sidebar containing a label (jLabel1), a button (jButton1), a dropdown menu (Item 1), and another button (jButton1).

I com he indicat abans, la fase 1 com tal no la he completat segons els criteris d'aquesta, si no que a mitat camí he passat a la Fase 2, així que ho acabaré d'explicar allí el que queda.

Fase 2

Mostrar text als elements del java.

En el punt en el que recientment hem creat la interfície, si executem el programa veurem com tots els botons i quadres de text tenen com a text la seva variable, o un nom predeterminat (algo com JTextField1). Nosaltres no ho volem així. Tindrem que crear a la vista els getters de tots els elements de la GUI.

A screenshot of an IDE window showing a Java source file. The code defines several getter methods for GUI components. The methods are organized into sections with comments: //COSES - VEHICLE, //TITOL, //LABELS - FORMULARI, //BOTO SUBMIT - FORMULARI, and //TEXTOS - FORMULARI. The methods return JLabel, JButton, and JTextField objects. The code is as follows:

```
543 }
544
545 //COSES - VEHICLE
546
547 //TITOL
548 public JLabel getVehicleLabel() {
549     return vehicleLabel;
550 }
551
552
553 //LABELS - FORMULARI
554 public JLabel getAfehirMarcaLabel() {
555     return afehirMarcaLabel;
556 }
557
558 public JLabel getAfehirModelLabel() {
559     return afehirModelLabel;
560 }
561
562 public JLabel getAfehirAnyLabel() {
563     return afehirAnyLabel;
564 }
565
566 public JLabel getAfehirNumeroLabel() {
567     return afehirNumeroLabel;
568 }
569
570
571 //BOTO SUBMIT - FORMULARI
572 public JButton getAfehirVehicleButton() {
573     return afehirVehicleButton;
574 }
575
576
577 //TEXTOS - FORMULARI
578 public JTextField getAfehirMarcaText() {
579     return afehirMarcaText;
580 }
581
582 public JTextField getAfehirModelText() {
583     return afehirModelText;
584 }
```

Una vegada creats, ja els podem utilitzar. En el meu cas, ho estic fent al fitxer del controlador: He creat un metode **defecteText** que carrega dades per defecte a la taula amb un metode que hem creat al model (que mostraré després) a part de tot el text que mostren els botons,etc.

Aquest metode serà cridat al principi de l'execució del controlador, fent que la nostra taula no sigui tant fea.

```

49 private void defecteText() {
50     /**
51      * VEHICLE
52      */
53
54     //Vehicles per defecte
55     model.insertarVehicle("Mazda", "RX-7 FC", 1989, 6);
56     model.insertarVehicle("Unity", "RX-7 FC", 1986, 8);
57     model.insertarVehicle("Nissan", "Skyline GTR R32", 1991, 22);
58     model.insertarVehicle("Toyota", "Corolla Trueno AE86", 1986, 86);
59     model.insertarVehicle("Nissan", "Silvia S15", 1998, 66);
60
61     //Conductors per defecte
62     model.insertarConductor("Pepe", "Viyuela", 45, 6589);
63     model.insertarConductor("Paul", "Walker", 47, 2254);
64     model.insertarConductor("Ian", "Lewis", 24, 222);
65     model.insertarConductor("Frank", "Williams", 53, 1);
66     model.insertarConductor("Alex", "Cañizares", 21, 1574);
67
68
69     ///////////////////////////////////////////////////
70
71     //TITOL PANEL VEHICLE
72     view.getVehicleLabel().setText("VEHICLES");
73
74     //CAIXES DE TEXT - FORMULARI
75     view.getAfegirMarcaText().setSize(0, 0);
76     view.getAfegirMarcaText().setText("Marca Placeholder");
77     view.getAfegirModelText().setText("Model Placeholder");
78     view.getAfegirAnyText().setText("9999");
79     view.getAfegirNumeroText().setText("999");
80
81     //FILTRE
82     view.getFiltrarVehiclesCombobox().removeAllItems();
83     view.getFiltrarVehiclesCombobox().addItem("Ordenar per numero");
84     view.getFiltrarVehiclesCombobox().addItem("Ordenar per marca");
85
86     //LABELS - FORMULARI
87     view.getAfegirMarcaLabel().setText("Marca Vehicle");
88     view.getAfegirModelLabel().setText("Model Vehicle");
89     view.getAfegirAnyLabel().setText("Any Vehicle");
90     view.getAfegirNumeroLabel().setText("Numero Vehicle");

```

Llistat Vehicles

NUMERO_VEHICLE	MODEL_VEHICLE	ANY_VEHICLE	MARCA_VEHICLE	
6	RX-7 FC	1989	Mazda	VEHICLES <div style="margin-top: 10px;"> <div style="border: 1px solid #ccc; padding: 2px; width: 100px; margin-bottom: 5px;">Eliminar fila!</div> <div style="border: 1px solid #ccc; padding: 2px; width: 100px; margin-bottom: 5px;">Ordenar per numero ▼</div> <div style="border: 1px solid #ccc; padding: 2px; width: 100px;">Editar!</div> </div>
8	RX-7 FC	1986	Unity	
22	Skyline GTR R32	1991	Nissan	
66	Silvia S15	1998	Nissan	
86	Corolla Trueno AE86	1986	Toyota	

Marca Vehicle
Any Vehicle

Afegir registre

Model Vehicle
Numero Vehicle

Marca Vehicle
Any Vehicle

Model Vehicle
Numero Vehicle

Inserir dades a la taula

Per a inserir dades a la taula del java primer tindrem que crear els objectes que aniran en aquesta taula.

Aquí a la fase 1 serien simples objectes de tipus Array. Com jo he començat amb la fase 2, aquests objectes s'han convertit en Col·leccions de tipus TreeSet (he elegit TreeSet sobre ArrayList perquè permet ser ordenat de forma més senzilla, encara que porti alguns problemes més tard).

Crearé 2 col·leccions, una per a les dades ordenades de una forma (data) i una altra per a les dades ordenades d'una altra forma (dataOrd). Ara dona igual l'ordenació, el que volem és que guardin informació. Les deus seràn de tipus <VEHICLE>, però de moment passem de la "dataOrd"

Crearem un getter per a cada.

I crearem el mètode **insertarVehicle**, que servirà per a insertar vehicles com el seu nom indica. L'hi passarem 4 paràmetres: Una marca, un model, un any i un número (Str, Str, int, int).

Aquest mètode el que farà és un .add sobre la col·lecció per defecte de un nou objecte de Tipus **Vehicle**, el qual tindrà una marca, model...

```
//Vehicles
private Collection<Vehicle> data = new TreeSet<>();
private Collection<Vehicle> dataOrd = new TreeSet<>(new VehicleOrdenatMarca());

public Collection<Vehicle> getData() {
    return data;
}

public Collection<Vehicle> getDataOrd() {
    return dataOrd;
}

public void insertarVehicle(String marca, String model, int any, int numero) {
    data.add(new Vehicle(marca, model, any, numero));
}
```

Però...quin tipus <VEHICLE>? El que crearem a la nova classe de java **Vehicle.java** dins al paquet del model. Aquí dins definirem que tindrà l'objecte (els seus paràmetres)

També farem els getters i setters dels paràmetres.

```
public Vehicle(String _4_marcaVehicle, String _2_modelVehicle, int _3_anyVehicle, int _1_numeroVehicle) {
    this._4_marca_Vehicle = _4_marcaVehicle;
    this._2_model_Vehicle = _2_modelVehicle;
    this._3_any_Vehicle = _3_anyVehicle;
    this._1_numero_Vehicle = _1_numeroVehicle;
}
```

```
start Page x Controller.java x Model.java x Vehicle.java x Conductor.java x M
Source History
6 L */
7 public class Vehicle implements Comparable<Vehicle> {
8
9     private String _4_marca_Vehicle;
10    private String _2_model_Vehicle;
11    private int _3_any_Vehicle;
12    private int _1_numero_Vehicle;
13
14    public String get4_marca_Vehicle() {
15        return _4_marca_Vehicle;
16    }
17
18    public void set4_marca_Vehicle(String _4_marcaVehicle) {
19        this._4_marca_Vehicle = _4_marcaVehicle;
20    }
21
22    public String get2_model_Vehicle() {
23        return _2_model_Vehicle;
24    }
25
26    public void set2_model_Vehicle(String _2_modelVehicle) {
27        this._2_model_Vehicle = _2_modelVehicle;
28    }
29
30    public int get3_any_Vehicle() {
31        return _3_any_Vehicle;
32    }
33
34    public void set3_any_Vehicle(int _3_anyVehicle) {
35        this._3_any_Vehicle = _3_anyVehicle;
36    }
37
38    public int get1_numero_Vehicle() {
39        return _1_numero_Vehicle;
40    }
41
42    public void set1_numero_Vehicle(int _1_numeroVehicle) {
43        this._1_numero_Vehicle = _1_numeroVehicle;
44    }
45
```


I ara ja estariem llestos per a introduir dades a la taula, amb 1 senzill pas: Cridar al metode insertarVehicle al controlador i posar-li algunes dades dins.

```
private void defecteText() {  
    /**  
     * VEHICLE  
     */  
  
    //Vehicles per defecte  
    model.insertarVehicle("Mazda", "RX-7 FC", 1989, 6);  
    model.insertarVehicle("Unity", "RX-7 FC", 1986, 8);  
    model.insertarVehicle("Nissan", "Skyline GTR R32", 1991, 22);  
    model.insertarVehicle("Toyota", "Corolla Trueno AE86", 1986, 86);  
    model.insertarVehicle("Nissan", "Silvia S15", 1998, 66);  
}
```

Però ara només hem afegit les dades, les tenim que mostrar.

Al controlador crearem una variable de tipus TableColumn, de nom a elegir.

```
private int filaSegona;  
private TableColumn tc;
```

Crearem un metode **carregarTaula** per tal de carregar les dades de la col·lecció a la Jtable. Per a aixó utilitzarem un metode **loadTable**, el qual ha estar proveït a nosaltres per el nostre professor.

A aquest metode l'hi passes el tipus d'objecte sobre el que estas treballant (<Vehicle>), les dades actuals introduïdes a la col·lecció (model.getData()) que hem creat anteriorment), la taula del java sobre la que vols posar l'informació, i la classe de la classe sobre la que estem treballant (Vehicle.class).

```
public void carregarTaulaVehicle() {  
    tc = Utils.<Vehicle>loadTable(model.getData(), view.getJTaulaVehicles(), Vehicle.class, true, true);  
}
```

Carregues aquest metode al inici de l'execució del controlador i màgia, ja tens dades a la taula.

Llistat Vehicles			
NUMERO_VEHICLE	MODEL_VEHICLE	ANY_VEHICLE	MARCA_VEHICLE
6	RX-7 FC	1989	Mazda
8	RX-7 FC	1986	Unity
22	Skyline GTR R32	1991	Nissan
66	Silvia S15	1998	Nissan
86	Corolla Trueno AE86	1986	Toyota

Però aquestes dades son dades per defecte, que posem hardcoded dins al metode defecteText(); .Ara volem donar-li us al formulari de creació de registres que hem creat.

El que farém es agregar un listener al botó d'afegir vehicles. El listener detectarà que es presioni el botó, i executarà un codi en conseqüencia. El codi serà aquest:

```
//afegirVehicle
view.getAfegirVehicleButton().addActionListener(e -> {
    model.insertarVehicle(view.getAfegirMarcaText().getText(),
        view.getAfegirModelText().getText(),
        Integer.parseInt(view.getAfegirAnyText().getText()),
        Integer.parseInt(view.getAfegirNumeroText().getText())
    );
    carregarTaulaVehicleActual();
});
});
```

El que farà serà utilitzar el metode **insertarVehicle** que hem creat anteriorment, al qual com a parametres l'hi passarem el contingut que hi hagi dins a les caselles de text emplenables del formulari.

Abans d'introduir algunes dades amb el formulari →

NUMERO_VEHICLE	MODEL_VEHICLE	ANY_VEHICLE	MA
6	RX-7 FC	1989	Mazda
8	RX-7 FC	1986	Unity
22	Skyline GTR R32	1991	Nissan
66	Silvia S15	1998	Nissan
86	Corolla Trueno AE86	1986	Toyota

Marca Vehicle

Any Vehicle

Afegir registre

Model Vehicle

Numero Vehicle

Desrpes d'introduir algunes dades amb el formulari →

NUMERO_VEHICLE	MODEL_VEHICLE	ANY_VEHICLE	MARCA_VEH
6	RX-7 FC	1989	Mazda
8	RX-7 FC	1986	Unity
22	Skyline GTR R32	1991	Nissan
66	Silvia S15	1998	Nissan
86	Corolla Trueno AE86	1986	Toyota
999	Model Placeholder	9999	Marca Placeholder

Marca Vehicle

Any Vehicle

Afegir registre

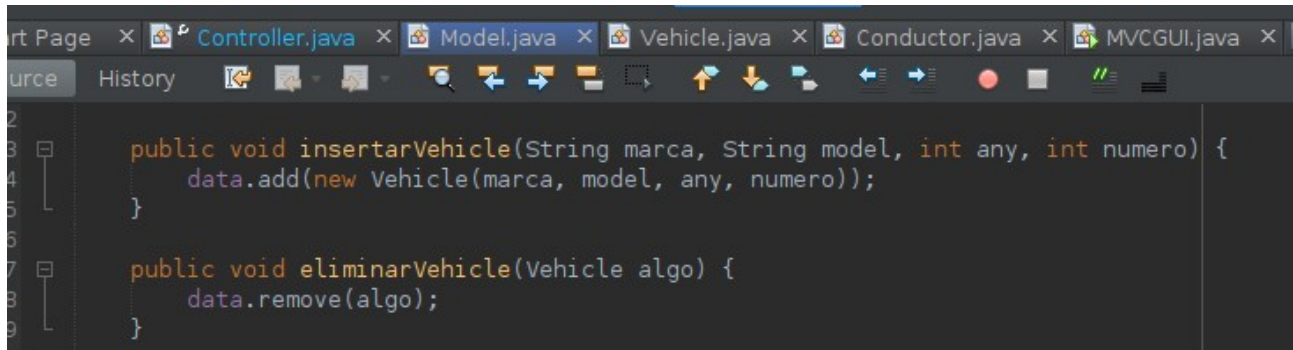
Model Vehicle

Numero Vehicle

Eliminar dades

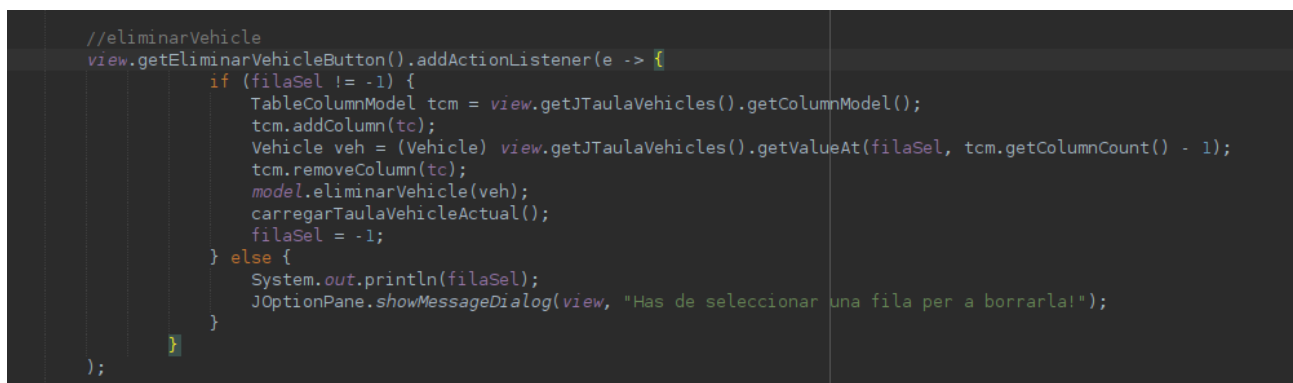
Per a eliminar dades començarem seguint els mateixos passos que abans:

-Crear un metode al model, el qual borri el objecte de la taula que l'hi passem



```
Controller.java x Model.java x Vehicle.java x Conductor.java x MVCGUI.java x
Source History
2
3 public void insertarVehicle(String marca, String model, int any, int numero) {
4     data.add(new Vehicle(marca, model, any, numero));
5 }
6
7 public void eliminarVehicle(Vehicle algo) {
8     data.remove(algo);
9 }
```

-Afegir un listener al controlador que detecti quan es presiona el botó d'eliminar un registre.



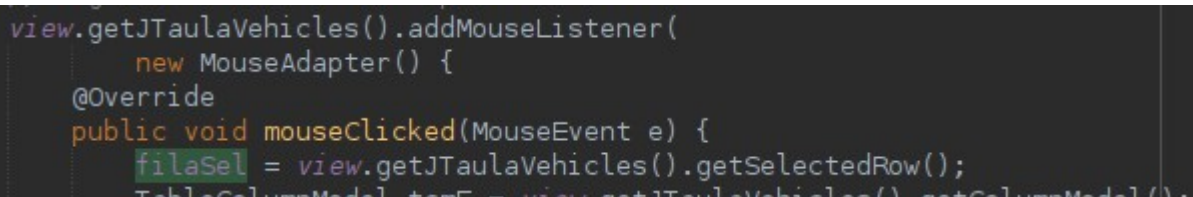
```
//eliminarVehicle
view.getEliminarVehicleButton().addActionListener(e -> {
    if (filaSel != -1) {
        TableColumnModel tcm = view.getJTaulaVehicles().getColumnModel();
        tcm.addColumn(tc);
        Vehicle veh = (Vehicle) view.getJTaulaVehicles().getValueAt(filaSel, tcm.getColumnCount() - 1);
        tcm.removeColumn(tc);
        model.eliminarVehicle(veh);
        carregarTaulaVehicleActual();
        filaSel = -1;
    } else {
        System.out.println(filaSel);
        JOptionPane.showMessageDialog(view, "Has de seleccionar una fila per a borrarla!");
    }
});
```

Aquí el problema es com elegir quina fila borrem eliminar nosaltres. Per a això implementarem una variable al controlador, **filaSel**, que ens indicarà quina es la fila que tenim seleccionada actualment.

```
private int filaSel = -1;
```

El que farem es crear un altre listener, que detecti qual elegim una fila a la taula. Quan elegim una, passa el parametre d'aquesta a la variable filaSel, la qual guardarà el numero de columna a la que estem actualment.

Si, es podria fer directament el metode **.getSelectedRow** però guardar-ho en una variable sempre es mes comode.



```
view.getJTaulaVehicles().addMouseListener(
    new MouseAdapter() {
        @Override
        public void mouseClicked(MouseEvent e) {
            filaSel = view.getJTaulaVehicles().getSelectedRow();
            TableColumnModel tcm = view.getJTaulaVehicles().getColumnModel();
```

I dins al codi del listener per a eliminarVehicle, la clau està aquí. Guardar en un objecte de tipus vehicle el registre localitzat a (getValueAt(filaSel, tcm.getColumnCount() - 1)).

Una vegada fet aixó, executarem el metode **eliminarVehicle** sobre el nou objecte, tornarem a carregar la taula, i indicarem que la fila sel·leccionada actual sigui -1 (que no sel·leccioni res després de borrar un registre, per a evitar problemes).

```
Vehicle veh = (Vehicle) view.getJTaulaVehicles().getValueAt(filaSel, tcm.getColumnCount() - 1);
tcm.removeColumn(tc);
model.eliminarVehicle(veh);
carregarTaulaVehicleActual();
filaSel = -1;
```

Hem de tenir que amb aquesta forma de fer-ho estarem eliminant l'ultima fila sel·leccionada, així que compte en no apretar el botó per accident.

Editar Vehicles

Al contrari que amb les operacions de CRUD anteriors, per a editar els vehicles no farem us d'un metode creat al model, ho farem tot directe al controlador.

La clau d'editar un vehicle...es editar-lo. Podriem agafar un vehicle, borrar-lo, i despres tornar-lo a crear amb les dades que nosaltres volem...però no seria el mateix vehicle com tal. Tenim que modificar les dades d'un vehicle sense borrar-lo.

Per a això farem ús de 2 listeners, que realment podria ser només un porque el primer es mes estètic que cap altra cosa.

El primer listener

Aquest s'activarà al clicar sobre una fila de la taula. Guardarà el numero de la fila que hem sel·leccionat a la variable mostrada anteriorment (si, el codi mostrat anteriorment forma part d'aquest listener), i a partir d'aquesta variable treurem el vehicle localitzat en aquesta fila.

I el que afrem es emplenar els camps de text **del formulari d'edició** amb les dades de la fila que anem sel·leccionant en cada moment. Així mirant directament al formulari sabrem quin vehicle estem editant en cada moment.

```
//VEHICLE
//editarVehicle
//Afegir text dinamic a l'apartat de edit
view.getJTaulaVehicles().addMouseListener(
    new MouseAdapter() {
        @Override
        public void mouseClicked(MouseEvent e) {
            filaSel = view.getJTaulaVehicles().getSelectedRow();
            TableColumnModel tcmE = view.getJTaulaVehicles().getColumnModel();
            tcmE.addColumn(tc);
            System.out.println(filaSel);
            Vehicle vehE = (Vehicle) view.getJTaulaVehicles().getValueAt(filaSel, tcmE.getColumnCount() - 1);
            System.out.println(String.valueOf(vehE));
            view.getEditorNumeroText().setText(String.valueOf(vehE.get1_numero_Vehicle()));
            view.getEditorAnyText().setText(String.valueOf(vehE.get3_any_Vehicle()));
            view.getEditorModelText().setText(vehE.get2_model_Vehicle());
            view.getEditorMarcaText().setText(vehE.get4_marca_Vehicle());
            tcmE.removeColumn(tc);
        }
    }
);
```

Si ara sel·leccionem el vehicle *Nissan Silvia S15* els camps de text del formulari d'edició (el de la dreta) mostraran l'informació d'aquest objecte, el que estem editant actualment.

NUMERO_VEHICLE	MODEL_VEHICLE	ANY_VEHICLE	MARCA_VEHICLE
6	RX-7 FC	1989	Mazda
8	RX-7 FC	1986	Unity
22	Skyline GTR R32	1991	Nissan
66	Silvia S15	1998	Nissan
86	Corolla Trueno AE86	1986	Toyota

VEHICLES
 Eliminar fila!
 Ordenar per numero
 Editar!

Marca Vehicle: Any Vehicle: **Afegir registre**

Marca Vehicle: Any Vehicle:
 Model Vehicle: Numero Vehicle:

Marca Vehicle: Any Vehicle:
 Model Vehicle: Numero Vehicle:

Si sel·leccionem un altre, s'actualitzarà automàticament.

NUMERO_VEHICLE	MODEL_VEHICLE	ANY_VEHICLE	MARCA_VEHICLE
6	RX-7 FC	1989	Mazda
8	RX-7 FC	1986	Unity
22	Skyline GTR R32	1991	Nissan
66	Silvia S15	1998	Nissan
86	Corolla Trueno AE86	1986	Toyota

VEHICLES
 Eliminar fila!
 Ordenar per numero
 Editar!

Marca Vehicle: Any Vehicle: **Afegir registre**

Marca Vehicle: Any Vehicle:
 Model Vehicle: Numero Vehicle:

Marca Vehicle: Any Vehicle:
 Model Vehicle: Numero Vehicle:

Segon listener: botó d'editar

El que farém es analitzar si estem sel·leccionant una fila. Si no n'estem sel·leccionant cap ens mostrarà un missatge conforme no hem sel·leccionat cap fila per a editar (el mateix que he fet al de eliminarVehicle), i si en tenim una de sel·leccionada ens executarà el codi de edició. Aquest consistirà en agafar el vehicle de la columna sel·leccionada i canviar-li els valors individualment amb els metodes `.set` que nosaltres hem creat anteriorment a la classe **Vehicle** al fitxer **Vehicle.java**.

```

view.getEditarVehicleButton().addActionListener(e -> {
    if (filaSel != -1) {
        TableColumnModel tcm = view.getJTaulaVehicles().getColumnModel();
        tcm.addColumn(tc);
        Vehicle veh = (Vehicle) view.getJTaulaVehicles().getValueAt(filaSel, tcm.getColumnCount() - 1);
        veh.set1_numero_Vehicle(Integer.parseInt(view.getEditarNumeroText().getText()));
        veh.set2_model_Vehicle(view.getEditarModelText().getText());
        veh.set3_any_Vehicle(Integer.parseInt(view.getEditarAnyText().getText()));
        veh.set4_marca_Vehicle(view.getEditarMarcaText().getText());
        carregarTaulaVehicleActual();
        tcm.removeColumn(tc);
        filaSel = -1;
    } else {
        System.out.println(filaSel);
        JOptionPane.showMessageDialog(view, "Has de seleccionar una fila per a editarla");
    }
});

```

Prova conforme no ens deixa editar si no sel·leccionem cap registre.

The screenshot shows the 'Llistat Vehicles' application. It features a table with four columns: NUMERO_VEHICLE, MODEL_VEHICLE, ANY_VEHICLE, and MARCA_VEHICLE. The table contains five rows of vehicle data. To the right of the table is a 'VEHICLES' section with buttons for 'Eliminar fila!', 'Ordenar per numero' (with a dropdown arrow), and 'Editar!'. Below the table are input fields for 'Marca Vehicle' (with a placeholder), 'Any Vehicle' (with the value '9999'), and a button 'Afegir registre'. Another set of input fields for 'Model Vehicle' (with a placeholder) and 'Numero Vehicle' (with the value '999') is also present. A 'Message' dialog box is open in the foreground, displaying the message 'Has de seleccionar una fila per a editarla' and an 'OK' button.

NUMERO_VEHICLE	MODEL_VEHICLE	ANY_VEHICLE	MARCA_VEHICLE
6	RX-7 FC	1989	Mazda
8	RX-7 FC	1986	Unity
22	Skyline GTR R32	1991	Nissan
66	Silvia S15	1998	Nissan
86	Corolla Trueno AE86	1986	Toyota

Prova d'edició d'un vehicle.

This screenshot shows the same application as the previous one, but with the 'Editar!' button in the 'VEHICLES' section highlighted. The table of vehicles remains the same. The input fields for 'Marca Vehicle' and 'Any Vehicle' are still '9999', and 'Model Vehicle' and 'Numero Vehicle' are still '999'. The 'Afegir registre' button is also visible.

NUMERO_VEHICLE	MODEL_VEHICLE	ANY_VEHICLE	MARCA_VEHICLE
6	RX-7 FC	1989	Mazda
8	RX-7 FC	1986	Unity
22	Skyline GTR R32	1991	Nissan
66	Silvia S15	1998	Nissan
2	Supra	1994	Toyota

Filtrar vehicles

Els requisits de la fase 2 ens demana implementar un sistema de filtre/ordenació de les dades. Així que aixó es el que he fet.

He posat una llista desplegable (JComboBox) on elegirem per que volem ordenar, si per numero de vehicle o marca de vehicle (de menor a major tant el numero com la marca, ordenant-se aquesta ultima alfabeticament).

This screenshot shows a close-up of the 'VEHICLES' section. It displays the 'Eliminar fila!' button, the 'Ordenar per numero' dropdown menu, and the expanded list of options: 'Ordenar per numero' and 'Ordenar per marca'.

El primer serà crear un nou listener per al ComboBox. Aquest funciona de la següent forma: el primer registre a la llista té un index de 0, el segon un 1, el tercer un 2...i així consecutivament.

Quan el index estigui a 0 (el valor per defecte), guardarem que estem al menú aquest en una variable al controlador, que controlarà en quin menú estem constantment. El listener detectasi canviem de filtre, i procedeix a canviar la variable (indicant al index que ens trobem actualment) i torna a carregar la taula.

Però...que fa aquest metode de **carregarTaulaVehicleActual**?

```
//FILTRE
view.getFiltrarVehiclesCombobox().addItemListener(e -> {
    if (view.getFiltrarVehiclesCombobox().getSelectedIndex() == 0) {
        colVehicleActual=0;
        carregarTaulaVehicleActual();
    }
    if (view.getFiltrarVehiclesCombobox().getSelectedIndex() == 1) {
        colVehicleActual=1;
        carregarTaulaVehicleActual();
    }
});
```

Aquest metode executarà un altre metode depenent de a quin filtre estem. Si estem al filtre per defecte, carregarà la taula normalment ordenada amb el filtre per defecte (que després el mostraré). Si no executarà el segon metode, que carregarà la taula però ordenada de una altra forma.

```
public void carregarTaulaVehicleActual(){
    if (colVehicleActual==0) {
        carregarTaulaVehicle();
    } else {
        carregarTaulaVehicleOrdenada();
    }
}
```

```
public void carregarTaulaVehicle() {
    tc = Utils.<Vehicle>loadTable(model.getData(), view.getJTaulaVehicles(), Vehicle.class, true, true);
}
public void carregarTaulaVehicleOrdenada() {
    model.getDataOrd().addAll(model.getData());
    tc = Utils.<Vehicle>loadTable(model.getDataOrd(), view.getJTaulaVehicles(), Vehicle.class, true, true);
}
```

Ara mirem com s'ordenen les dades.

El tipus de col·lecció **TreeSet** permet ordenar les dades. Per a ordenarles per defecte tindrem que fer canvis a la classe **Vehicle**. Tindrem que implementar el metode **Comparable**.

```
Page x Controller.java x Model.java x Vehicle.java x Cond
ce History
/*
 * To change this license header, choose License Headers in Project Properties
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package model;

import java.util.Comparator;
import javax.swing.JTable;
import javax.swing.table.DefaultTableModel;

/**
 *
 * @author profe
 * Vehicle
 */
public class Vehicle implements Comparable<Vehicle> {
```

I ens farà implementar el metode **compareTo**. Aquí es on indicarem com s'ordenaran les dades.

En el meu cas, he fet que s'ordenin per numero i per marca. Per dos? Si, porque TreeSet no pot ordenar registres duplicats. Es dir, que si ordenem per numeroVehicle però n'hi han 2 amb el mateix numero, no mostrarà 1 dels dos registres...i si afegim mes vehicles amb numeros repetits aquestos tampoc es mostraran. Així que la solució esta en comparar amb varios camps, així que si troba un numero repetit compari la vehicleMarca amb l'altre registre.

```
Start Page x Controller.java x Model.java x Vehicle.java x Conductor.java x MVCGUI.java x View.java x
Source History
63
64
65
66
67 @Override
68 public int compareTo(Vehicle o) {
69 // return this._1_numero_Vehicle-o._1_numero_Vehicle;
70 return Comparator.comparing(Vehicle::get1_numero_Vehicle).thenComparing(Vehicle::get4_marca_Vehicle)
71 .compare(this, o);
72 }
```

I aquí quan creem la col·lecció amb el tipus d'objecte <Vehicle>, ja estarem indicant que les dades es carreguin així.

```
//Vehicles
private Collection<Vehicle> data = new TreeSet<>();
```

```
public Collection<Vehicle> getData() {
    return data;
}
```


NUMERO_VEHICLE	
6	RX-7 FC
8	RX-7 FC
22	Skyline
66	Silvia S1
86	Corolla

I per al segon filtro una segona col·lecció.

Guardarem el nou TreeSet en un atribut diferent, i al TreeSet l'hi indicarem que volem carregar les dades de una altra forma, com ho tenim fet a **VehicleOrdenatMarca**, el qual crearem mes abaix.

```
//Vehicles
private Collection<Vehicle> data = new TreeSet<>();
private Collection<Vehicle> dataOrd = new TreeSet<>(new VehicleOrdenatMarca());

public Collection<Vehicle> getData() {
    return data;
}

public Collection<Vehicle> getDataOrd() {
    return dataOrd;
}
```

I aquí es on indicarem com ordenarem les dades amb la segona opció. En aquest cas ordenarem per marca, i si hi ha duplicat que ordeni per model.

```
class VehicleOrdenatMarca implements Comparator<Vehicle> {

    @Override
    public int compare(Vehicle o1, Vehicle o2) {
        // return o1.get4_marca_Vehicle().compareTo(o2.get4_marca_Vehicle());
        int p;
        p = o1.get4_marca_Vehicle().compareTo(o2.get4_marca_Vehicle());
        if (p != 0) {
            return p;
        }
        return o1.get2_model_Vehicle().compareTo(o2.get2_model_Vehicle());
    }
}
```

I ara ja podrem veure la llista ordenada per marca.

NUMERO_VEHICLE	MODEL_VEHICLE	ANY_VEHICLE	MARCA_VEHICLE
6	RX-7 FC	1989	Mazda
66	Silvia S15	1998	Nissan
22	Skyline GTR R32	1991	Nissan
86	Corolla Trueno AE86	1986	Toyota
8	RX-7 FC	1986	Unity

VEHICLES

 ▾

Marca Vehicle
 Any Vehicle

Model Vehicle
 Numero Vehicle

Segona col·lecció

Per a la fase 1 hem de tenir un POJO creat fet en un simple array, en el meu cas era vehicle.

A la fase 2, teniem que passar aquest array a una col·lecció i, a part, crear-ne un altre. En el meu cas, he creat **CONDUCTORS**.

Hi han diferents formes de fer un segon “pojo” en MVC. En aplicacions web, hem vist que per cada “pagina” que vulguem tindre hauriem de tenir 2 controladors diferents, 2 vistes diferents...en java no ho faré així.

Mantindré un sol fitxer de controlador, vista i model. Així si, el fitxer que teniem de la classe “Vehicle” el duplicarem i l’adaptarem a la nova classe. Aquests seràn els canvis.

Al model tindrem que crear les col·leccions per a la nou objecte “Conductor” (si, 2. Recordem que també tindrem que tornar a fer els filtres ja que els anteriors no serveixen).

Tambè tindrem que crear un metode per insertar i eliminar conductors, ja que al tenir camps diferents al del objecte anterior no podem realment aprofitar el mateix codi.

Tambè implementarem aquí el que serà el segon filtre, al igual que en Vehicles.

```
//Conductor
private Collection<Conductor> dataConductor = new TreeSet<>();
private Collection<Conductor> dataOrdConductor = new TreeSet<>(new ConductorOrdenatNom());

public Collection<Conductor> getDataConductor() {
    return dataConductor;
}

public Collection<Conductor> getDataOrdConductor() {
    return dataOrdConductor;
}

public void insertarConductor(String nom, String cognom, int edat, int id) {
    dataConductor.add(new Conductor(nom, cognom, edat, id));
}

public void eliminarConductor(Conductor algo) {
    dataConductor.remove(algo);
}

}

class ConductorOrdenatNom implements Comparator<Conductor> {

    @Override
    public int compare(Conductor o1, Conductor o2) {
```

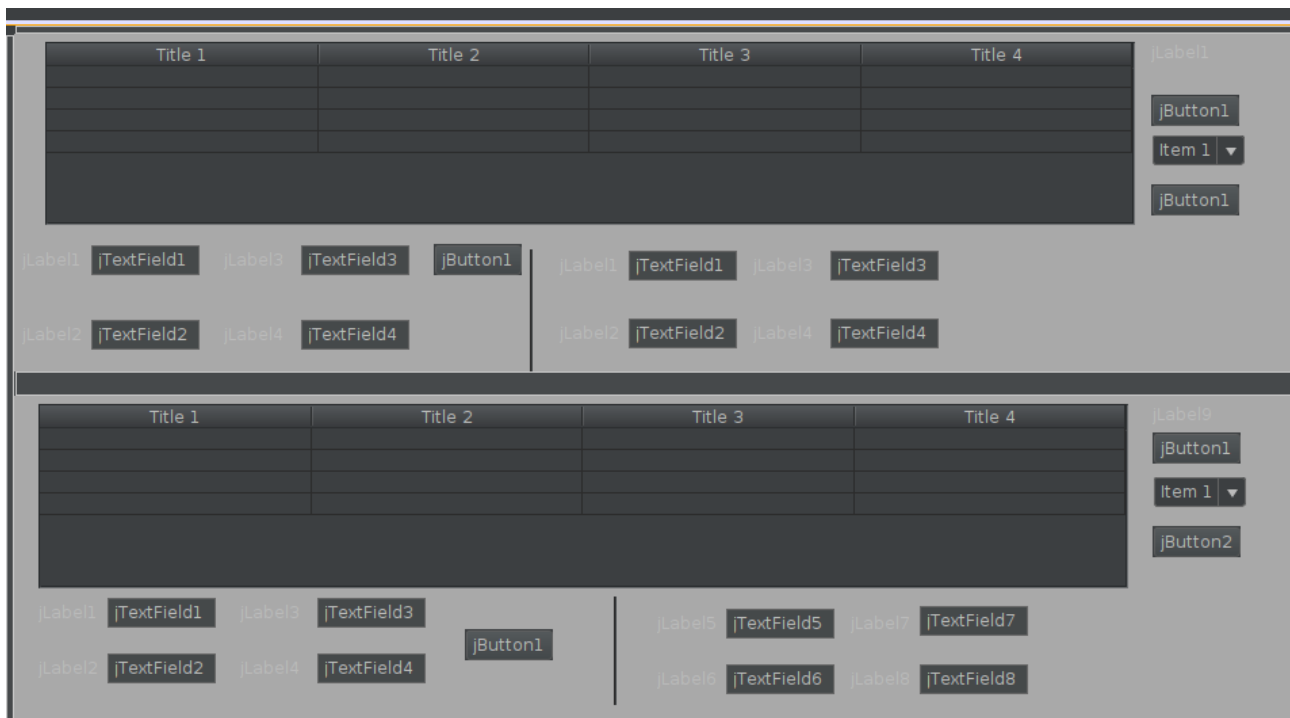
Crearem un nou arxiu per a la segona classe, “Conductor.java”. Al igual que a la primera, crearem els setters i getters de els parametres del objecte i el crearem com tal.

```
Start Page x Controller.java x Model.java x Vehicle.java x Conductor.java x MVCGUI.java x View.java x
Source History
16 L */
17 public class Conductor implements Comparable<Conductor> {
18
19     private String _4_nom_Conductor;
20     private String _2_cognom_Conductor;
21     private int _3_edat_Conductor;
22     private int _1_id_Conductor;
23
24     public String get4_nom_Conductor() {
25         return _4_nom_Conductor;
26     }
27
28     public void set4_nom_Conductor(String _4_nomConductor) {
29         this._4_nom_Conductor = _4_nomConductor;
30     }
31
32     public String get2_cognom_Conductor() {
33         return _2_cognom_Conductor;
34     }
35
36     public void set2_cognom_Conductor(String _2_cognomConductor) {
37         this._2_cognom_Conductor = _2_cognomConductor;
38     }
39
40     public int get3_edat_Conductor() {
41         return _3_edat_Conductor;
42     }
43
44     public void set3_edat_Conductor(int _3_edatConductor) {
45         this._3_edat_Conductor = _3_edatConductor;
46     }
47
48     public int get1_id_Conductor() {
49         return _1_id_Conductor;
50     }
51
52     public void set1_id_Conductor(int _1_idConductor) {
53         this._1_id_Conductor = _1_idConductor;
54     }
55
56     public Conductor(String _4_nomConductor, String _2_cognomConductor, int _3_edatConductor, int _1_idConductor) {
57         this._4_nom_Conductor = _4_nomConductor;
58         this._2_cognom_Conductor = _2_cognomConductor;
```

I aquí també implementarem el filtre per defecte. En aquest cas filtrarem per la ID i si es duplica una ID per ID+nom. El segon filtre (el de dins al Model) filtrarà primer per nom, i si aquest esta duplicat per cognom.

```
@Override
public int compareTo(Conductor o) {
//     return this._1_id_Conductor - o._1_id_Conductor;
    return Comparator.comparing(Conductor::get1_id_Conductor).thenComparing(Conductor::get4_nom_Conductor)
        .compare(this, o);
}
```

A la vista també duplicarem la interfície. La tenim que adaptar al nou objecte, però com realment els he fet molt pareguts no hi ha molta diferencia entre ells.



I finalment el controlador. Aquí també es farà el mateix que amb l'altre objecte: inicialitzar els valors dels elements de la vista, crear mètodes per carregar la nova taula i afegir listeners per als nous botons i interaccions a la nova taula.

```
Start Page x Controller.java x Model.java x Vehicle.java x Conductor.java x MVCGUI.java x
Source History
110
111
112 /**
113  * Conductor
114  */
115 //TITOL PANEL VEHICLE
116 view.getConductorLabel().setText("CONDUCTORS");
117
118 //CAIXES DE TEXT - FORMULARI
119 //
120 view.getAfegirMarcaText().setSize(0, 0);
121 view.getAfegirNomConductorText().setText("Nom Placeholder");
122 view.getAfegirCognomConductorText().setText("Cognom Placeholder");
123 view.getAfegirEdatConductorText().setText("999");
124 view.getAfegirIdConductorText().setText("99999");
125
126 //FILTRE
127 view.getFiltrarConductorCombobox().removeAllItems();
128 view.getFiltrarConductorCombobox().addItem("Ordenar per ID");
129 view.getFiltrarConductorCombobox().addItem("Ordenar per nom");
130
131 //LABELS - FORMULARI
132 view.getAfegirNomConductorLabel().setText("Nom Conductor");
133 view.getAfegirCognomConductorLabel().setText("Cognom Conductor");
134 view.getAfegirEdatConductorLabel().setText("Edat Conductor");
135 view.getAfegirIdConductorLabel().setText("ID Conductor");
136
137 //BOTÓ SUBMIT - FORMULARI
138 view.getAfegirConductorButton().setText("Afegir registre");
139
140 //FORMULARI ELIMINAR - TEXT DEFECTE
141 view.getEliminarConductorButton().setText("Eliminar fila!");
142
143
144 view.getEditarConductorButton().setText("Editar!");
145 view.getEditarNomConductorLabel().setText("Nom Conductor");
146 view.getEditarEdatConductorText().setText("999");
147 view.getEditarCognomConductorLabel().setText("Cognom Conductor");
148 view.getEditarNomConductorText().setText("Placeholder");
149 view.getEditarCognomConductorText().setText("Placeholder");
150 view.getEditarEdatConductorLabel().setText("Edat Conductor");
151 view.getEditarIdConductorLabel().setText("ID Conductor");
152 view.getEditarIdConductorText().setText("99999");
```

```

public void carregarTaulaConductor() {
    tc = Utils.<Conductor>loadTable(model.getDataConductor(), view.getJTaulaConductor(), Conductor.class, true, true);
}

public void carregarTaulaConductorOrdenada() {
    model.getDataOrdConductor().addAll(model.getDataConductor());
    tc = Utils.<Conductor>loadTable(model.getDataOrdConductor(), view.getJTaulaConductor(), Conductor.class, true, true);
}

public void carregarTaulaConductorActual() {
    if (colConductorActual==0) {
        carregarTaulaConductor();
    } else {
        carregarTaulaConductorOrdenada();
    }
}
}

```

Aquí cal afegir un incís, i és que també tindrem que crear noves variables per a coses com “Fila de taula seleccionada”, ja que no volem sobre escriure qualsevol cosa que estigui passant en aquesta taula, o en una futura nova taula.

```

////
//CONDUCTOR
////
//editarVehicle
//Afegir text dinamic a l'apartat de edit
view.getJTaulaConductor().addMouseListener(
    new MouseAdapter() {
        @Override
        public void mouseClicked(MouseEvent e) {
            filaSelCond = view.getJTaulaConductor().getSelectedRow();
            TableColumnModel tcmCondE = view.getJTaulaConductor().getColumnModel();
            tcmCondE.addColumn(tc);
            System.out.println(filaSel);
            Conductor condE = (Conductor) view.getJTaulaConductor().getValueAt(filaSelCond, tcmCondE.getColumnCount() - 1);
            System.out.println(String.valueOf(vehE));
            view.getEditorIdConductorText().setText(String.valueOf(condE.get1_id_Conductor()));
            view.getEditorEdatConductorText().setText(String.valueOf(condE.get3_edat_Conductor()));
            view.getEditorCognomConductorText().setText(condE.get2_cognom_Conductor());
            view.getEditorNomConductorText().setText(condE.get4_nom_Conductor());
            tcmCondE.removeColumn(tc);
        }
    }
);
view.getEditorConductorButton().addActionListener(e -> {
    System.out.println(filaSel);
    if (filaSelCond != -1) {
        TableColumnModel tcm = view.getJTaulaConductor().getColumnModel();
        tcm.addColumn(tc);
        System.out.println(filaSel);
        Conductor cond = (Conductor) view.getJTaulaConductor().getValueAt(filaSelCond, tcm.getColumnCount() - 1);
        System.out.println(veh.toString());
        cond.set1_id_Conductor(Integer.parseInt(view.getEditorIdConductorText().getText()));
        cond.set2_cognom_Conductor(view.getEditorCognomConductorText().getText());
        cond.set3_edat_Conductor(Integer.parseInt(view.getEditorEdatConductorText().getText()));
        cond.set4_nom_Conductor(view.getEditorNomConductorText().getText());
        tcm.removeColumn(tc);
        carregarTaulaConductorActual();
        filaSelCond = -1;
    }
});

```

```

//eliminarConductor
view.getEliminarConductorButton().addActionListener(e -> {
    System.out.println(filaSel);
    if (filaSelCond != -1) {
        TableColumnModel tcm = view.getJTaulaConductor().getColumnModel();
        tcm.addColumn(tc);
        System.out.println(filaSel);
        Conductor cond = (Conductor) view.getJTaulaConductor().getValueAt(filaSelCond, tcm.getColumnCount() - 1);
        System.out.println(veh.toString());
        tcm.removeColumn(tc);
        model.eliminarConductor(cond);
        carregarTaulaConductorActual();
        filaSelCond = -1;
    } else {
        System.out.println(filaSelCond);
        JOptionPane.showMessageDialog(view, "Has de seleccionar una fila per a borrarla!");
    }
});

view.getFiltrarConductorCombobox().addItemListener(e -> {
    if (view.getFiltrarConductorCombobox().getSelectedIndex() == 0) {
        colConductorActual=0;
        carregarTaulaConductorActual();
    }
    if (view.getFiltrarConductorCombobox().getSelectedIndex() == 1) {
        colConductorActual=1;
        carregarTaulaConductorActual();
    }
});
}

```

I així quedaria el programa després d’haver afegit aquesta segona taula.

Llistat Vehicles

NUMERO_VEHICLE	MODEL_VEHICLE	ANY_VEHICLE	MARCA_VEHICLE
6	RX-7 FC	1989	Mazda
8	RX-7 FC	1986	Unity
22	Skyline GTR R32	1991	Nissan
66	Silvia S15	1998	Nissan
86	Corolla Trueno AE86	1986	Toyota

VEHICLES

Eliminar fila!

Ordenar per numero ▾

Editar!

Marca Vehicle

Marca Placeholder

Any Vehicle

9999

Afegir registre

Marca Vehicle

Placeholder

Any Vehicle

9999

Model Vehicle

Model Placeholder

Numero Vehicle

999

Model Vehicle

Placeholder

Numero Vehicle

999

ID_CONDUCTOR	COGNOM_CONDUCTOR	EDAT_CONDUCTOR	NOM_CONDUCTOR
1	Williams	53	Frank
222	Lewis	24	Ian
1574	Cañizares	21	Alex
2254	Walker	47	Paul
6589	Viyuela	45	Pepe

CONDUCTORS

Eliminar fila!

Ordenar per ID ▾

Editar!

Nom Conductor

Nom Placeholder

Edat Conductor

999

Afegir registre

Nom Conductor

Placeholder

Edat Conductor

999

Cognom Conductor

Cognom Placeholder

ID Conductor

99999

Cognom Conductor

Placeholder

ID Conductor

99999

FASE 3

Aquí la documentació ja serà menys extensa, ja que es simplement afegir nova informació al projecte ja existent.

Anem per parts sobre els requisits. Primer de tot tenim que afegir algun camp que sigui un array o una cadena de caràcters. En el meu cas me he inclinat cap a un array de Strings.

L'implementaré sobre vehicle, i guardarà els 3 sponsors que tindrà cada vehicle (sponsor=anunciant/publicitat).

Crearem un nou atribut de classe, juntament amb els seus getters/setters. El getter serà una mica diferent de la resta, ja que m'interessa mostrar l'informació de forma que sigui visible a la taula.

```
*/
public class Vehicle implements Comparable<Vehicle> {

    private String[] _5_sponsors_Vehicle;
    private String _4_marca_Vehicle;
    private String _2_model_Vehicle;
    private int _3_any_Vehicle;
    private int _1_numero_Vehicle;
    public Collection<Conductor> _6_cond = new TreeSet<>();

    public String get5_sponsors_Vehicle() {
        String res = "";
        for (int i = 0; i < _5_sponsors_Vehicle.length; i++) {
            res = res + _5_sponsors_Vehicle[i].toString() + ",";
        }
        return res;
    }

    public void set5_sponsors_Vehicle(String[] _5_sponsors_Vehicle) {
        this._5_sponsors_Vehicle = _5_sponsors_Vehicle;
    }
}
```

Evidentment modificarem l'objecte «Vehicle» per a que aquest tingui el nou camp.

```
public Vehicle(String _4_marcaVehicle, String _2_modelVehicle, int _3_anyVehicle, int _1_numeroVehicle, String[] _5_sponso
    this._5_sponsors_Vehicle = _5_sponsors_Vehicle;
    this._4_marca_Vehicle = _4_marcaVehicle;
    this._2_model_Vehicle = _2_modelVehicle;
    this._3_any_Vehicle = _3_anyVehicle;
    this._1_numero_Vehicle = _1_numeroVehicle;
}
```


El metode insertar vehicle també es tindrà que canviar per a acomodar el nou parametre. El de “Model.insertar” forma part de les classes generiques, que explicaré més tard.

```
public void insertarVehicle(String marca, String model, int any, int numero, String[] sponsors) {
    Vehicle ve = new Vehicle(marca, model, any, numero, sponsors);
    Model.insertar(ve, data);
    Model.insertar(ve, dataOrd);
}
```

I per no plenar tot el document amb captures d’aixó, mencionar que també s’ha canviat la vista (per tal de que els usuaris puguin introduir vehicles amb sponsors) i el controlador per tal de posar el nou text de exemple, i al insertar el vehicle i mostrar la taula.

NUMERO_VEHICLE	MODEL_VEHICLE	ANY_VEHICLE	MARCA_VEHICLE	SPONSORS_VEHICLE	COND
6	RX-7 FC	1989	Mazda	Pirelli, Repsol, SPAR...	[]
8	RX-7 FC	1986	Unity	Pirelli, Repsol, SPAR...	[]
22	Skyline GTR R32	1991	Nissan	Pirelli, Repsol, SPAR...	[]

Eliminar fila!

Ordenar per numero ▼

Editar!

Marca Vehicle

Placeholder

Any Vehicle

1900

Afegir registre

Marca Vehicle

Placeholder

Any Vehicle

9999

Model Vehicle

Placeholder

Numero Vehicle

99

Sponsor 1

Exemple

Sponsor 2

Exemple

Sponsor 3

Exemple

El segon requisit es l’us de metodes generics en les operacions de CRUD.

En el meu cas n’he fet 2, una per a insertar i l’altra per a eliminar. Per a editar no he fet cap metode generic ja que l’edició la faig directament sobre el controlador.

Aquí es pot veure el metode genèric que he creat per tal de insertar dades a una col·lecció. A aquest l’hi passarem un objecte de qualsevol tipus (vehicle, conductor, etc..) i una col·lecció.

Al metode insertarVehicle es pot veure el seu ús.

```
//metode generic per a insertar dades --- també serveix per a conductor
public static <T> void insertar(T a, Collection<T> col) {
    col.add(a);
}

public void insertarVehicle(String marca, String model, int any, int numero, String[] sponsors) {
    Vehicle ve = new Vehicle(marca, model, any, numero, sponsors);
    Model.insertar(ve, data);
    Model.insertar(ve, dataOrd);
}
```

I gràcies a que és un metode generic també el podem utilitzar a conductor sense cap problema.

```
public void insertarConductor(String nom, String cognom, int edat, int id, Vehicle vehicle_Conductor) {
    Conductor co = new Conductor(nom, cognom, edat, id, vehicle_Conductor);
    Model.insertar(co, dataConductor);
    Model.insertar(co, dataOrdConductor);
}
```


I el mateix per a eliminar, vehicles en aquest cas.

```
//metode generic per a eliminar dades
public static <T> void eliminar(T a, Collection<T> col) {
    col.remove(a);
}

public void eliminarVehicle(Vehicle algo) {
    Model.eliminar(algo, data);
    Model.eliminar(algo, dataOrd);
}
```

Tercer requisit es l'us d'alguna expressió regular per comprovar la validesa d'algun camp.

Fet sobre el listener de “afegirVehicle” al controlador, aquest if combinat amb expressions regulars comprova que el camp “numero_vehicle” tingui 2 o 1 dígit.

```
//Exemple de validesa utilitzant expressions regulars
if (view.getAfegirNumeroText().getText().matches("\\d{2}") || view.getAfegirNumeroText().getText().matches("\\d{1}")) {
    String[] sponsors_vehicle = {view.getAfegirSponsor1Text().getText(), view.getAfegirSponsor2Text().getText(), view.getAfegirSponsor3Text().getText()};
}
```

FASE 4

Aquí demana dues coses. La primera es “delegar excepcions”, cosa que faré a la fase 5 degut al fet de que encara no tenim cap checked.

La segona es el tractament de camps normal i corrent, per a que l’usuari no introdueixi el que l’hi doni la gana.

Ho he fet tant al listener de afegir un vehicle com al de afegir un conductor. En aquest cas poso d’exemple el de vehicle, fent que es tinguin que complir els següents requisits:

- Cap parametre pot estar buit al afegir un vehicle.
- El numero del vehicle ha de tenir 1 o 2 dígit (expressió regular). Al haver-ho fet en una expressió regular de la forma en que ho he fet, també peta si introduïm una lletra i no un dígit (2x1 en tractament d’excepció).
- L’any del vehicle ha de ser un any vàlid per a un vehicle (entre l’any 1900 i el 2030).
- L’any no pot ser una lletra, ha de ser un numero (tractat en lo try-catch).

```
//afegirVehicle
view.getAfegirVehicleButton().addActionListener(e -> {

    if (view.getAfegirNumeroText().getText().isBlank()
        || view.getAfegirMarcaText().getText().isBlank()
        || view.getAfegirModelText().getText().isBlank()
        || view.getAfegirAnyText().getText().isBlank()) {
        JOptionPane.showMessageDialog(view, "Hi ha algun camp buit. No pot haver-hi cap camp buit!");
    } else {
        //Exemple de validesa utilitzant expressions regulars
        if (view.getAfegirNumeroText().getText().matches("\\d{2}") || view.getAfegirNumeroText().getText().matches(
            String[] sponsors_vehicle = {view.getAfegirSponsor1Text().getText(), view.getAfegirSponsor2Text().getTe
            try {
                if (Integer.parseInt(view.getAfegirAnyText().getText()) < 1900
                    || Integer.parseInt(view.getAfegirAnyText().getText()) > 2030) {
                    JOptionPane.showMessageDialog(view, "El any ha de ser valid (entre 1900 i 2030)");
                } else {
                    model.insertarVehicle(view.getAfegirMarcaText().getText(),
                        view.getAfegirModelText().getText(),
                        Integer.parseInt(view.getAfegirAnyText().getText()),
                        Integer.parseInt(view.getAfegirNumeroText().getText()),
                        sponsors_vehicle
                    );
                }
            } catch (NumberFormatException x) {
                JOptionPane.showMessageDialog(view, "El any ha de ser un ANY (en numeros, no escrit)");
            }
            actualizarComboBoxCond();
            carregarTaulaVehicleActual();
            carregarTaulaVehicleActual();
        } else {
            JOptionPane.showMessageDialog(view, "El numero del vehicle ha de ser inferior a 100! I no pot ser"
                + " una paraula, lletres, etc..");
        }
    }
})
```

FASE 5

Aquí hi han més coses que veure que a la Fase 4, així que anem per parts.

Primer de tot el guardar les dades en un fitxer per tal de que no es borri al tancar el programa.

He creat 2 mètodes al “Model.java”, un per a cada tipus de col·lecció. Aquests mètodes requeriran d’un string, que serà el nom del fitxer on es guardarà l’informació...aquí 2 coses:

1. Els usuaris poden introduir el nom del fitxer: Això no he fet. He provat de tocar una mica el codi per tal de que funcioni i només dona problemes. A part, no ho veig una millora molt útil, la veritat.
2. El recomanat era que es guardessin les 2 col·leccions (o 3 o 4 en un futur) en un mateix fitxer. Jo no se com fer-ho, així que hi haurà un fitxer per col·lecció.

Tambè hi ha el de delegació de excepcions de la Fase 4 (throws AlgoException)

El que guardarem als fitxers es l’informació de la col·lecció sencera, no de cada vehicle/conductor individual.

```
// GUARDAR LES DADES DELS VEHICLES A UN FITXER
public void saveVehicle(String filename) throws FileNotFoundException, IOException {
    ObjectOutputStream out = null;

    try {
        out = new ObjectOutputStream(new BufferedOutputStream(new FileOutputStream(filename)));
        out.writeObject(data);
    } finally {
        if (out != null) {
            out.close();
        }
    }
}

// GUARDAR LES DADES DELS CONDUCTORS A UN FITXER
public void saveConductor(String filename) throws FileNotFoundException, IOException {
    ObjectOutputStream out = null;

    try {
        out = new ObjectOutputStream(new BufferedOutputStream(new FileOutputStream(filename)));
        out.writeObject(dataConductor);
    } finally {
        if (out != null) {
            out.close();
        }
    }
}
```

Aquests dos metodes els cridarem al “Controlador”. Aquest “listener” es un listener que detecta quan es tanca el programa, es dir que quan es tanca el programa executa el codi que té dins.

Tot això es fa que fer dins d’un try catch, ja que al delegar les excepcions al model les tenim que tractar en algun moment al utilitzar-les, en aquest cas al controlador.

```
//GUARDAR LES DADES A UN FITXER QUAN ES TANCA EL PROGRAMA
Runtime.getRuntime().addShutdownHook(new Thread() {
    @Override
    public void run() {
        //MODEL DE PROVA 2
        for (int i = 0; i < model.getData().size(); i++) {
            try {
                model.saveVehicle(nomArxiuV);
                model.saveConductor(nomArxiuC);
            } catch (IOException ex) {
                System.out.println("Catch de excepcio de quan es crea el fitxer per primera vegada");
            }
        }
    }
});
```

I aquestos dos altres metodes seràn per a llegir les dades dels fitxers, també creats al Model.

```
}

//CARREGAR LES DADES DELS VEHICLES QUE ESTAN GUARDATS A UN FITXER
public void loadVehicle(String filename) throws FileNotFoundException, IOException, ClassNotFoundException {
    ObjectInputStream in = new ObjectInputStream(new FileInputStream(filename));
    try {
        data.addAll((TreeSet<Vehicle>) in.readObject());
    } finally {
        if(in != null) in.close();
    }
}

//CARREGAR LES DADES DELS CONDUCTORS QUE ESTAN GUARDATS A UN FITXER
public void loadConductor(String filename) throws FileNotFoundException, IOException, ClassNotFoundException {
    ObjectInputStream in = new ObjectInputStream(new FileInputStream(filename));
    try {
        dataConductor.addAll((TreeSet<Conductor>) in.readObject());
    } finally {
        if(in != null) in.close();
    }
}
```

Aquestos metodes els cridaré a un metode que he creat al controlador anomenat “carregarDades()”, el qual es cridarà al principi del controlador com es pot veure a la imatge.

Perquè no ho faig directament al controlador? Al fer la delegació d’excepcions tenim que fer alguns try-catch, que poden arribar a confondre una mica el codi. Així, i el fet de que el metode “controlador()” el hauriem de tenir el més net possible, fent ús de la programació modular.

```

    }

    public void carregarDades() {
        try {
            model.loadVehicle(nomArxiuV);
        } catch (IOException ex) {
            System.out.println("a");
        } catch (ClassNotFoundException ex) {
            System.out.println("a");
        }

        try {
            model.loadConductor(nomArxiuC);
        } catch (IOException ex) {
            System.out.println("a");
        } catch (ClassNotFoundException ex) {
            System.out.println("a");
        }
    }

    private void controlador() {

        try {
            preguntarPassword();
        } catch (FileNotFoundException ex) {
            System.out.println("No s'ha trobat algun fitxer, jo que se xdd");
        }

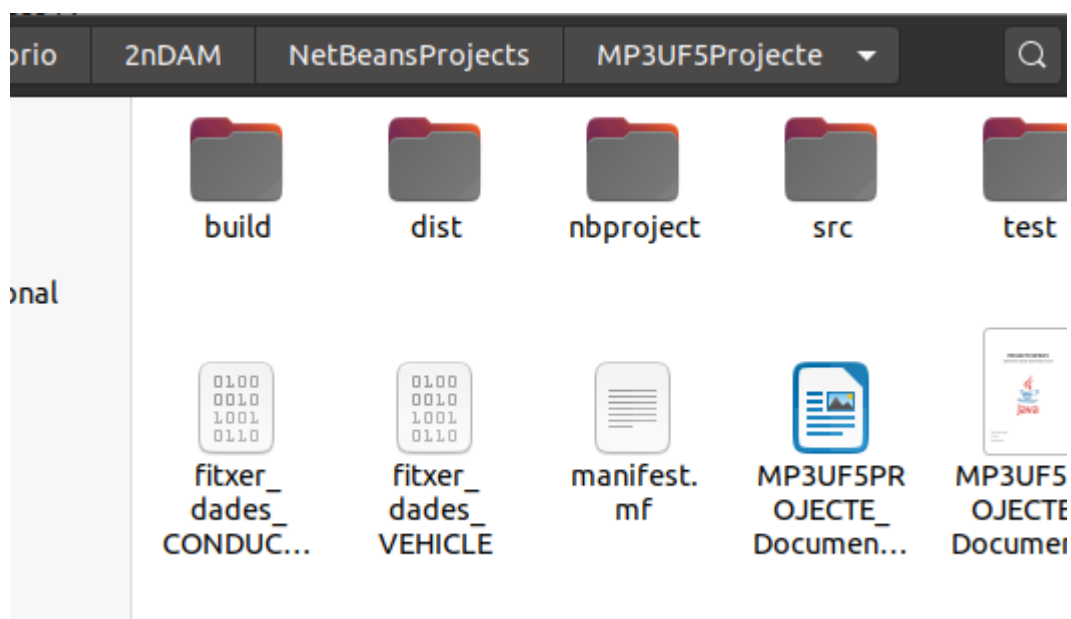
        //Codi que inicilitza la vista
        view.setVisible(true);

        //CARREGAR DADES A LES COL·LECCIONS A PARTIR DELS FITXERS
        carregarDades();
    }

```

Una vegada tenim fer aixó, cada vegada que tanquessim el programa es guardarán els canvis que haguessim fet a les taules i aquestes es carregarán al iniciar el programa.

Nota: En la configuració actual aquestos fitxers es creen a l'arrel del projecte.



Password (Contrasenya). Es demana que al iniciar el programa demani una password, la qual estarà guardada en un lloc aleatori de un fitxer de configuració binari.

Per a portar-ho a cap he fet un altre metode que s'executarà al principi del controlador, "preguntarPassword()".

Consisteix en el següent:

- Crearem un numero aleatori, que definirà la posició on es guardarà la password dins del fitxer. He limitat el rang per que a vegades creava arxius extremadament grans (+1.5GB).

- Crearem un nou fitxer amb el "File", aquí es on es guardarà la password.

- Dins del "RandomAccessFile", ens posicionarem al principi del fitxer (seek(0)) i guardarem la password (String secret) a la posició aleatoria definida anteriorment (offset).

- Una vegada creat el fitxer, executarem un "JOptionPane" que ens preguntarà sobre la password. La que introdueixi el usuari es guardarà en un string (choice).

- Farem un if comprovant si la password es la que es troba al fitxer o no. Si NO es correcta, mostra un missatge i surt del programa. Si SI es la correcta, entra al programa.

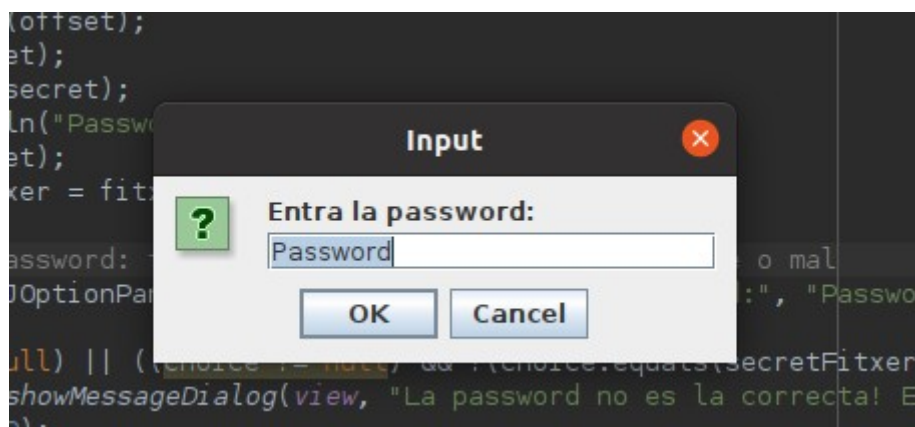
```
public void preguntarPassword() throws FileNotFoundException {
    //carregar la password al fitxer
    Random rn = new Random();
    long offset = rn.nextInt(1000 - 100) + 100;
    String secret = "701554";

    System.out.println(offset);
    File f = new File("secret.dat");
    try ( RandomAccessFile fitxer = new RandomAccessFile(f, "rw")) {
        fitxer.seek(0);
        fitxer.writeLong(offset);
        fitxer.seek(offset);
        fitxer.writeUTF(secret);
        System.out.println("Password carregada correctament");
        fitxer.seek(offset);
        String secretFitxer = fitxer.readUTF();

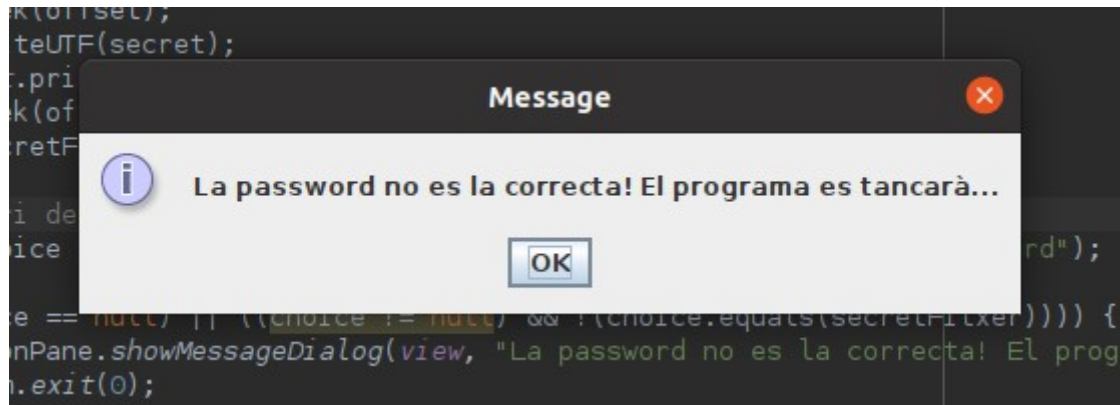
        //Formulari de password: trobarla al fitxer i mirar si està bè o mal
        String choice = JOptionPane.showInputDialog("Entra la password:", "Password");

        if ((choice == null) || ((choice != null) && !(choice.equals(secretFitxer)))) {
            JOptionPane.showMessageDialog(view, "La password no es la correcta! El programa es tancarà...");
            System.exit(0);
        }
    } catch (Exception e) {
        System.out.println("Algo dolent ha passat al carregar la password al fitxer!");
    }
}
```

Aquest es el missatge que surt al iniciar el programa



I aquest es el missatge si fallem la password.



El fitxer de la password, al igual que els fitxers de dades, també es guardarà a l'arrel del fitxer.

