DIARI D'UN REFACTORING

Neus Bravo Arias
CFGS: Desenvolupament d'aplicacions web
M5:UF2
9/05/2023

INDEX

INTRODUCCIÓ

El següent treball documentarà el procés de refacció d'un programa el qual serà creat des de zero. El programa a construir tracta sobre un sistema de gestió de lloguers. A partir dels lloguers generats pels clients, es calcularà els imports i bonificacions corresponents per cadascun. Utilitzarem, per començar, tres classes anomenades Vehicles, Lloguer i Clients, els lloguers fan referència als vehicles i els clients guarden la llista de lloguers contractats.
La refacció d'aquest projecte consisteix en, sense canviar ni afegir o treure funcionalitats, modificar el codi de manera que sigui més llegible, fàcil d'entendre i fàcil de modificar que el que era abans.
A mesura que avancem amb el programa, canviarem el codi constantment, anirem deixant constància d'això i documentarem totes les passes en aquestes pàgines.

DIAGRAMES

Diagrama de classes

Per donar pas a l'inici del projecte, comencem recordant que és un diagrama de classes i per a que serveix.
Aquests diagrams són una eina utilitzada per representar visualment les classes i les relacions entre elles dins d'un programa que es dur a terme mitjançant la programació orientada a objectes.
Cada classe es representarà amb una caixa amb el seu contingut i les relacions entre elles seràn les fletxes que les uneixen.

El següent diagrama fa referència a les tres classes inicials del programa de gestió de lloguers:

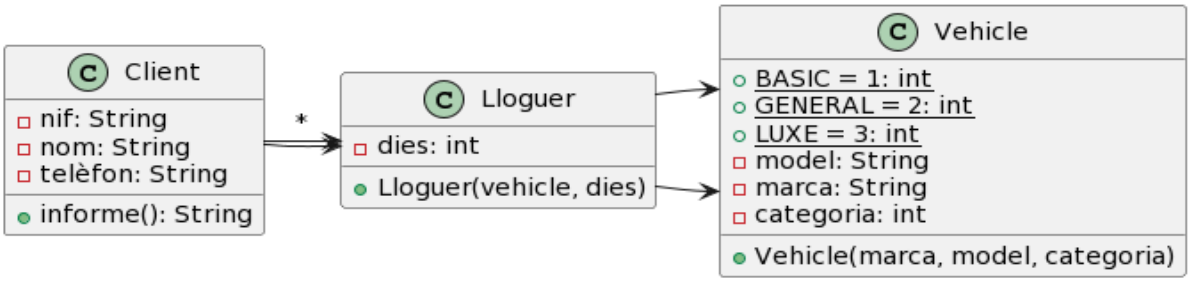
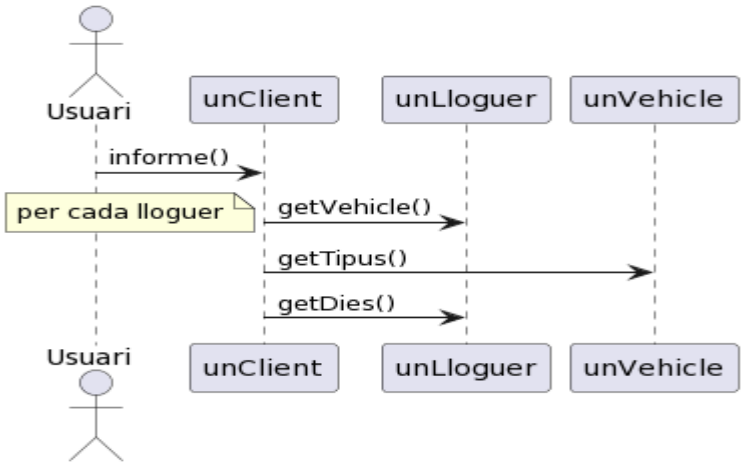


Diagrama de seqüència

Per altra banda, un diagrama de seqüència també representa visualment a les classes i les seves relacions, però aquesta vegada ho fa dins d'un escenari específic.
Es recreen les seqüències ocurrides entre les classes durant l'escenari, el que ajuda a entendre el funcionament i la lògica mateixa del programa.



EXTRACCIÓ DE MÈTODES

En aquesta nova secció començarem a documentar les primer passes de refacció del nostre codi després d'haver codificat les classes inicials a base de proves unitàries que nosaltres mateixos hem creat.
Aquest procés que ara explicaré és conegut com a ***extracció de mètode*** i té l'objectiu de descomposar un mètode tan llarg com el que nosaltres tractarem, Client.informe(), en parts més petites per tal de guanyar en quant a llegibilitat.

Aquest era el mètode originalment:

```
29      public String informe() {
30          double total = 0;
31          int bonificacions = 0;
32          String resultat = "Informe de lloguers del client " +
33              getNom() +
34              " (" + getNif() + ")\n";
35          for (Lloguer lloguer: lloguers) {
36              double quantitat = 0;
37              switch (lloguer.getVehicle().getCategoria()) {
38                  case Vehicle.BASIC:
39                      quantitat += 3;
40                      if (lloguer.getDies() > 3) {
41                          quantitat += (lloguer.getDies() - 3) * 1.5;
42                      }
43                      break;
44                  case Vehicle.GENERAL:
45                      quantitat += 4;
46                      if (lloguer.getDies() > 2) {
47                          quantitat += (lloguer.getDies() - 2) * 2.5;
48                      }
49                      break;
50                  case Vehicle.LUXE:
51                      quantitat += lloguer.getDies() * 6;
52                      break;
53              }
54
55              // afegeix lloguers freqüents
56              bonificacions ++;
57
58              // afegeix bonificació per dos dies de lloguer de Luxe
59              if (lloguer.getVehicle().getCategoria() == Vehicle.LUXE &&
60                      lloguer.getDies()>1 ) {
61                  bonificacions ++;
62              }
63
64              // composa els resultats d'aquest lloguer
65              resultat += "\t" +
66                  lloguer.getVehicle().getMarca() +
67                  " " +
68                  lloguer.getVehicle().getModel() + ": " +
69                  (quantitat * 30) + "€" + "\n";
70              total += quantitat * 30;
71          }
72
73          // afegeix informació final
74          resultat += "Import a pagar: " + total + "€\n" +
75              "Punts guanyats: " + bonificacions + "\n";
76          return resultat;
77      }
```

La proposta d'aquesta extracció era aconseguir treure la part de codi que tracta el *switch* i afegir-ho a un nou mètode anomenat ***quantitatPerLloguer().***
En aquest cas ha sigut altament senzill poder extreure el petit bloc sense tenir problemes amb les variables que es comparteixen al llarg de tot el mètode informe(), encara així, hem hagut d'ajustar dues variables com són quantitat , la qual hem declarat i inicialitzar una altra vegada al nou mètode, i lloguer, la qual hem passar per paràmetre ja que no comportava cap problema en aquest sentit.

El resultat de l'extracció ha estat el següent:
Client.informe() després de l'extracció:

```
29      public String informe() {
30          double total = 0;
31          int bonificacions = 0;
32          String resultat = "Informe de lloguers del client " +
33              getNom() +
34              " (" + getNif() + ")\n";
35          for (Lloguer lloguer: lloguers) {
36              double quantitat = quantitatPerLloguer(lloguer);
37
38              // afegeix lloguers freqüents
39              bonificacions ++;
40
41              // afegeix bonificació per dos dies de lloguer de Luxe
42              if (lloguer.getVehicle().getCategoria() == Vehicle.LUXE &&
43                      lloguer.getDies()>1 ) {
44                  bonificacions ++;
45              }
46
47              // composa els resultats d'aquest lloguer
48              resultat += "\t" +
49                  lloguer.getVehicle().getMarca() +
50                  " " +
51                  lloguer.getVehicle().getModel() + ": " +
52                  (quantitat * 30) + "€" + "\n";
53              total += quantitat * 30;
54          }
55
56          // afegeix informació final
57          resultat += "Import a pagar: " + total + "€\n" +
58              "Punts guanyats: " + bonificacions + "\n";
59          return resultat;
60      }
```

quantitatPerLloguer() com a nou mètode:

```
61      // ############# EXTRACCIÓ DE MÈTODE ###############
62      private double quantitatPerLloguer(Lloguer lloguer) {
63          double quantitat = 0;
64          switch (lloguer.getVehicle().getCategoria()) {
65              case Vehicle.BASIC:
66                  quantitat += 3;
67                  if (lloguer.getDies() > 3) {
68                      quantitat += (lloguer.getDies() - 3) * 1.5;
69                  }
70                  break;
71              case Vehicle.GENERAL:
72                  quantitat += 4;
73                  if (lloguer.getDies() > 2) {
74                      quantitat += (lloguer.getDies() - 2) * 2.5;
75                  }
76                  break;
77              case Vehicle.LUXE:
78                  quantitat += lloguer.getDies() * 6;
79                  break;
80          }
81          return quantitat;
82      }
```

MOVIMENT DE MÈTODES

El nou pas de la refacció a conèixer és el ***moviment de mètodes.***
Aquest procés consisteix en analitzar quins mètodes (o propietats) estan desubicats allà on els trobem, ja sigui perquè no fan ús de la seva classe per res o perquè encaixarien millor en altre lloc, i moure'ls d'una classe a una altra.
En el nostre cas hem vist que el mètode que prèviament hem extraigut, Client.quantitatPerLloguer(), no utilitza cap utilitat que l'ofereix la seva classe Client, en canvi una altre classe com Lloguer li encaixaria millor degut al context en el que es tracta.
La proposta aquesta vegada era moure aquest mètode de la classe Client a classe Lloguer fent els seus deguts canvis com: canvi de nom, obrir l'accés al mètode, modificar l'accés des de Client, etc.

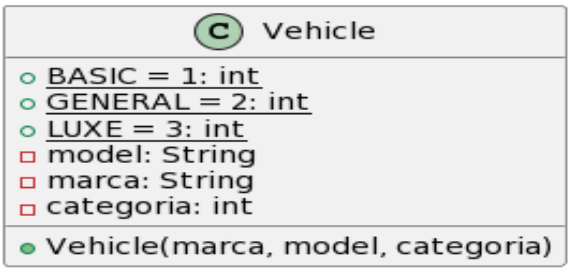
El resultat final ha estat el següent:

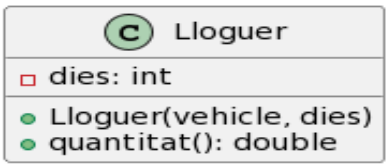
```
34      // METODES
35      // ############## MOVIMENT DE MÈTODE #############
36      public double quantitat() {
37          double quantitat = 0;
38          switch (this.getVehicle().getCategoria()) {
39              case Vehicle.BASIC:
40                  quantitat += 3;
41                  if (this.getDies() > 3) {
42                      quantitat += (this.getDies() - 3) * 1.5;
43                  }
44                  break;
45              case Vehicle.GENERAL:
46                  quantitat += 4;
47                  if (this.getDies() > 2) {
48                      quantitat += (this.getDies() - 2) * 2.5;
49                  }
50                  break;
51              case Vehicle.LUXE:
52                  quantitat += this.getDies() * 6;
53                  break;
54          }
55          return quantitat;
56      }
```

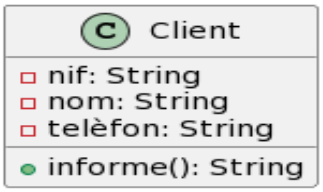
Com es pot entendre, hem eliminat el mètode de la classe Client i ho hem insertat en classe Lloguer com podem veure a la imatge d'adalt. Per facilitar-nos la vida, ja no fem ús d'un paràmetre sinó de la referencia this, la qual està cridant aquest mètode des de la classe Client.

En conseqüència d'aquest moviment, el diagrama de classes s'ha vist afectat, quedant finalment així:

ACTUALITZACIÓ DIAGRAMA DE CLASSES:







CAS PRÀCTIC DE REFACCIÓ: càlcul de bonificacions

En el següent apartat documentarem el procés pas per pas de la refacció que hem posat en pràctica amb el càlcul de bonificacions que trobem de manera desordenada al mètode Client.informe().
Seguint els passos que vam fer amb el càlcul de la quantitat per lloguer, començarem la refacció de les bonificacions amb l'extracció de mètodes.
L'objectiu d'aquest pas era aconseguir crear un mètode que calculi les bonificacions externament i retorni el valor a la variable dins del mètode informe().

L'estat del mètode Client.informe() després de l'extracció de les bonificacion va ser aquest:

```
29      public String informe() {
30          double total = 0;
31          int bonificacions = 0;
32          String resultat = "Informe de lloguers del client " +
33              getNom() +
34              " (" + getNif() + ")\n";
35          for (Lloguer lloguer: lloguers) {
36              bonificacions += bonificacionsDeLloguer(lloguer);
37              // composa els resultats d'aquest lloguer
38              resultat += "\t" +
39                  lloguer.getVehicle().getMarca() +
40                  " " +
41                  lloguer.getVehicle().getModel() + ": " +
42                  (quantitat(lloguer) * 30) + "€" + "\n";
43              total += quantitat(lloguer) * 30;
44          }
45
46          // afegeix informació final
47          resultat += "Import a pagar: " + total + "€\n" +
48              "Punts guanyats: " + bonificacions + "\n";
49          return resultat;
50      }
```

El mètode va resultar en això:

```
56      // ############## REFACCIÓ BONIFICACIONS ###################################
57      private int bonificacionsDeLloguer(Lloguer lloguer) {
58          int bonificacions = 1;
59          // afegeix bonificació per dos dies de lloguer de Luxe
60          if (lloguer.getVehicle().getCategoria() == Vehicle.LUXE &&
61                  lloguer.getDies()>1 ) {
62              bonificacions ++;
63          }
64          return bonificacions;
65      }
66  }
```

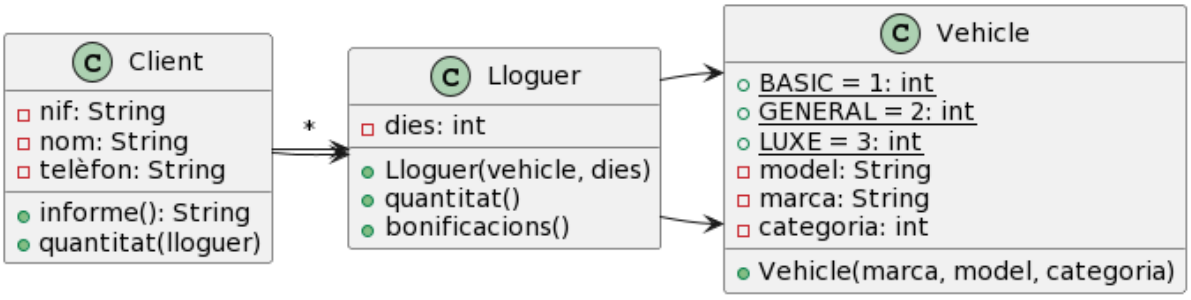
El següent pas consistia a concloure en si aquest mètode necessàriament ha d'estar a la classe Client. Necessita d'aquesta classe? La resposta va ser no, per tant fent una ullada al mètode i les crides que fa, es pot veure que encaixa millor a la classe Lloguer.
El segon que vam aprendre va ser moure mètodes i això és el que hem fet amb aquest. Client.bonificacionsDeLloguer() passa a ser part de la classe Lloguer i a nombrar-se simplement Lloguer.bonificacions(), on es torna un mètode públic i dinàmic i ja no necessita paràmetres i permet fer ús de this.

Ho eliminem de la classe Client i el resultat de la classe Lloguer és aquest:

```
58      // Moviment de mètode de càlcul de bonificacions
59      // Client - Lloguer
60      public int bonificacions() {
61          int bonificacions = 1;
62          // afegeix bonificació per dos dies de lloguer de Luxe
63          if (this.getVehicle().getCategoria() == Vehicle.LUXE &&
64                  this.getDies()>1 ) {
65              bonificacions ++;
66          }
67          return bonificacions;
68      }
```

Després d'això, hem de tornar a replantejar el diagrama de classes amb el resultat següent:

ACTUALITZACIÓ DIAGRAMA DE CLASSES:



ELIMINACIÓ DE VARIABLES TEMPORALS

El nou pas de refacció a conèixer és l'***eliminació de variables temporals*** les quals són variables que només permeten l'accés en els mètodes que s'utilitzen.
En aquest cas , i tenint en compte que el nostre mètode Client.informe() realitza tres tasques en el mateix bloc de codi, la idea serà separar en dos mètodes diferents els càlculs que es realitzen per l'import total i per les bonificacions, eliminant d'aquesta manera les variables temporals del mètode informe();

El resultat de l'eliminació de variable al mètode Client.informe() ha deixat el codi en aquest estat de puresa:

```
29      public String informe() {
30          String resultat = "Informe de lloguers del client " +
31              getNom() +
32              " (" + getNif() + ")\n";
33          for (Lloguer lloguer: lloguers) {
34              // composa els resultats d'aquest lloguer
35              resultat += "\t" +
36                  lloguer.getVehicle().getMarca() +
37                  " " +
38                  lloguer.getVehicle().getModel() + ": " +
39                  (quantitat(lloguer) * 30) + "€" + "\n";
40          }
41          // afegeix informació final
42          resultat += "Import a pagar: " + importTotal() + "€\n" +
43              "Punts guanyats: " + bonificacionsTotal() + "\n";
44          return resultat;
45      }
```

La variable temporal "total" s'ha vist transformada en aquest mètode:

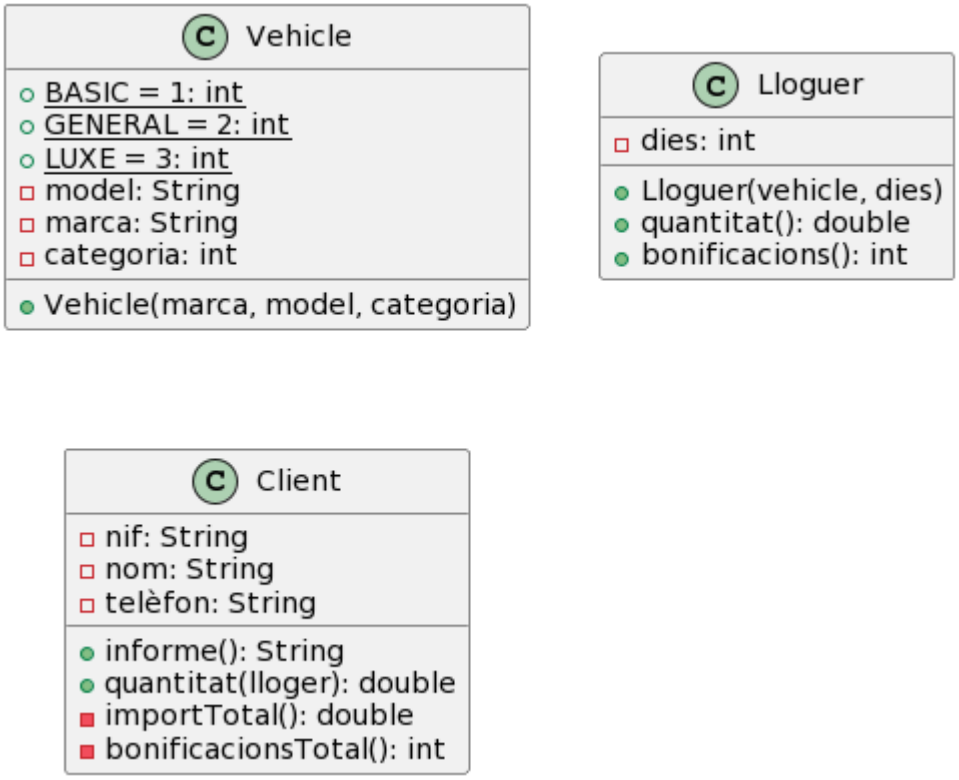
```
50      }
51      // ############## ELIMINACIÓ DE VARIABLES ################################
52      // Variable total
53      private double importTotal() {
54          double total = 0;
55          for (Lloguer lloguer: lloguers) {
56              total += quantitat(lloguer) * 30;
57          }
58          return total;
59      }
```

I les bonificaciones ara es calcularan en aquest altre:

```
60      // Variable bonificacions
61      private int bonificacionsTotal() {
62          int bonificacions = 0;
63          for (Lloguer lloguer: lloguers) {
64              bonificacions += lloguer.bonificacions();
65          }
66          return bonificacions;
67      }
68  }
```

En conseqüència als canvis efectuats en les classes, el diagrama de classes s'ha tornat a veure afectat fins quedar en aquest estat:

ACTUALITZACIÓ DIAGRAMA DE CLASSES:



UNA FUNCIONALITAT PER MÈTODE

Per terminar de polir el mètode Client.informe(), el següent pas serà afegir tres nous mètodes els quals optimitzaran el bloc de codi que genera la capçalera de l'informe, altre pels detall i finalment un pel peu.
Aquest pas és bastant senzill en el sentit que el codi és bastant clar en quina línea comença un bloc i en quina el següent.

El resultat pel mètode Client.informe() ha estat el següent després d'afegir els nous mètodes:

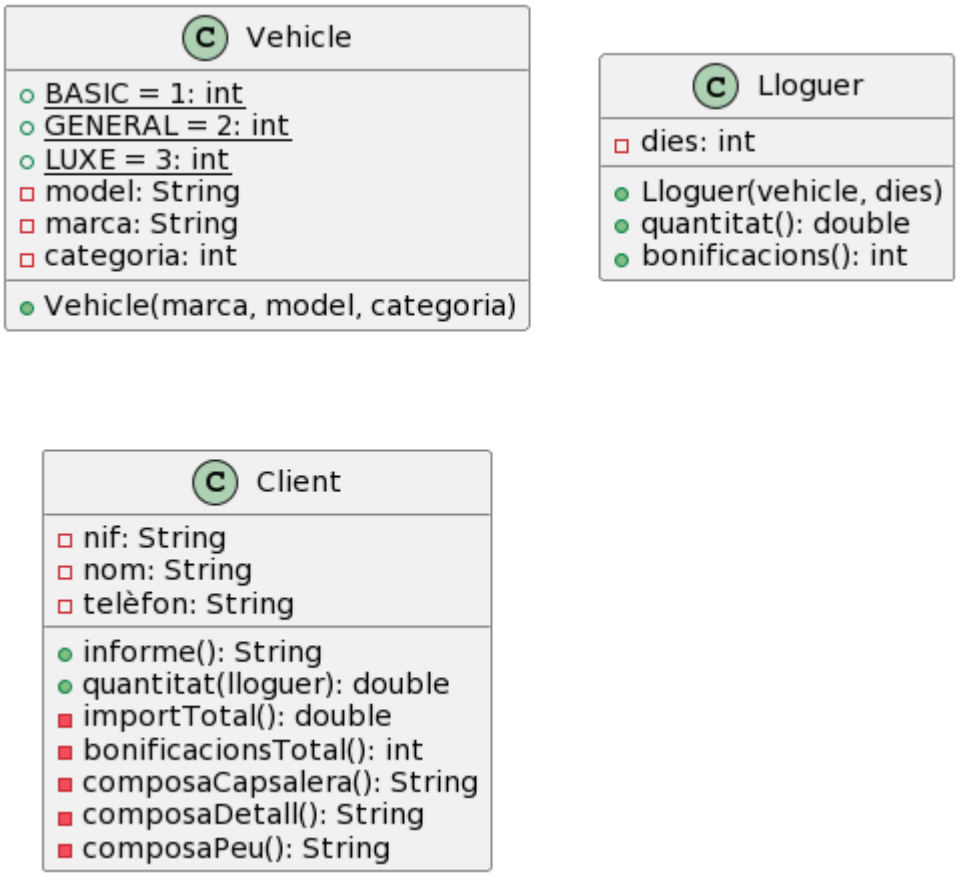
```
29      public String informe() {
30          return composaCapsalera() +
31             composaDetall() +
32             composaPeu();
33      }
```

El mètode que genera la capçalera, el dels detalls i el del peu són els següents per ordre:

```
57      // ############## METODES LLEUGERS ####################################
58      // construcció de capçalera
59      private String composaCapsalera() {
60          String cap = "Informe de lloguers del client " +
61           getNom() +
62           " (" + getNif() + ")\n";
63          return cap;
64      }
65      // composa detall
66      private String composaDetall() {
67          String resultat = "";
68          for (Lloguer lloguer: lloguers) {
69              // composa els resultats d'aquest lloguer
70              resultat += "\t" +
71                  lloguer.getVehicle().getMarca() +
72                  " " +
73                  lloguer.getVehicle().getModel() + ": " +
74                  (quantitat(lloguer) * 30) + "€" + "\n";
75          }
76          return resultat;
77      }
78      // composa peu
79      private String composaPeu() {
80          String resultat = "";
81          resultat += "Import a pagar: " + importTotal() + "€\n" +
82              "Punts guanyats: " + bonificacionsTotal() + "\n";
83          return resultat;
84      }
```

L'actualització del diagrama de classes amb tots aquest canvis ha estat la següent:

ACTUALITZACIÓ:



INFORME HTML: pas final

L'últim d'aquest projecte era crear el mètode final: Client.informeHTML().
Un mètode que mostra la informació que tenim a informe() però en un format HTML.

INCISO:
En plena correcció d'errors m'he trobat amb la redundància del mètode quantitat(). El qual estava repetit en la mateixa classe Client i en el moviment de mètode que vam fer del mètode quantitat de Client a Lloguer.
L'error era el següent:

✗	Client.java: S'espera que Client.informe() inclogui dues crides a Lloguer.quantitat() Client.java: [lin 61]: El lliurament no respecta els requeriments de l'enunciat.

Al tenir un mètode quantitat() dins la classe Client que bàsicament retornava un nombre que calculava a partir del mètode real que és Lloguer.quantitat() no estava cridant dues vegades al mètode real, sinó a l'altre.
Aquest error ha sigut arreglat simplement eliminant el mètode redundant i canviant les crides del mètode Client.informe().

Ara podem continuar amb la creació del mètode informeHTML().

Les proves unitàries en les que hem basat la construcció d'aquest mètode han estat aquestes:

```
161     // ########################################## INFORME HTML ##############################################
162     @Test
163     public void comprovaCapseleraHTML() {
164         Client demo = new Client("515906950", "Eugènia Salinas Roig", "93614214242");
165         demo.getLlogers().add(new Lloguer(new Vehicle("Seat", "600", 1), 2));
166         String esperat = "<p>Informe de lloguers del client <em>Eugènia Salinas Roig</em> (<strong>51590695Q</strong>)</p>\n";
167 
168         Assertions.assertEquals(esperat, demo.composaCapsaleraHTML());
169     }
170     @Test
171     public void comprovaDetallHTML() {
172         Client demo = new Client("515906950", "Eugènia Salinas Roig", "93614214242");
173         demo.getLlogers().add(new Lloguer(new Vehicle("Tata", "Vista", 1), 5));
174         demo.getLlogers().add(new Lloguer(new Vehicle("Seat", "600", 1), 2));
175         String esperat = "<table>\n" +
176                          "  <tr>\n" +
177                          "    <td><strong>Marca</strong></td>\n" +
178                          "    <td><strong>Model</strong></td>\n" +
179                          "    <td><strong>Import</strong></td>\n" +
180                          "  </tr>\n" +
181                          "  <tr><td>Tata</td><td>Vista</td><td>180.0€</td></tr>\n" +
182                          "  <tr><td>Seat</td><td>600</td><td>90.0€</td></tr>\n" +
183                          "</table>\n";
184 
185         Assertions.assertEquals(esperat, demo.composaDetallHTML());
186     }
187     @Test
188     public void comprovaPeuHTML() {
189         Client demo = new Client("515906950", "Eugènia Salinas Roig", "93614214242");
190         demo.getLlogers().add(new Lloguer(new Vehicle("Tata", "Vista", 1), 5));
191         String esperat = "<p>Import a pagar: <em>180.0€</em></p>\n" +
192                          "<p>Punts guanyats: <em>1</em></p>";
193 
194         Assertions.assertEquals(esperat, demo.composaPeuHTML());
195     }
196     @Test
197     public void comprovaInformeHTMLDemo() {
198         Client demo = new Client("515906950", "Eugènia Salinas Roig", "93614214242");
199         String esperat = demo.informeHTML();
200 
201         Assertions.assertEquals(esperat, demo.informeHTML());
202     }
203 }
```

Tenim una prova per cada secció de l'informe i una demo pel GestorLite.creaClientDemo(). En base a això, he generat els mètodes respectius amb aquest resultat:

Mètode recolector de seccions:

```
87      // ###################### informeHTML() ##########################
88      public String informeHTML() {
89          return composaCapsaleraHTML() + composaDetallHTML() + composaPeuHTML();
90      }
```

Capçalera de l'informe:

```
91      public String composaCapsaleraHTML() {
92          return String.format("<p>Informe de lloguers del client <em>%s</em> (<strong>%s</strong>)</p>%n", this.getNom(), this.getNif());
93      }
```

Detalls de l'informe:

```
94      public String composaDetallHTML() {
95          Locale.setDefault(Locale.US);
96          String doubleFormat;
97          String resultat = "<table>\n  <tr>\n    <td><strong>Marca</strong></td>\n    <td><strong>Model</strong></td>\n    <td><strong>Im
98          for (Lloguer lloguer : lloguers) {
99              resultat += String.format("  <tr><td>%s</td><td>%s</td><td>%s€</td></tr>\n", lloguer.getVehicle().getMarca(), lloguer.getVeh
    alculQuantitatsPerLloguer() * EUROS_PER_UNITAT_DE_COST)); }
100         resultat += "</table>\n";
101         return resultat;
102     }
```

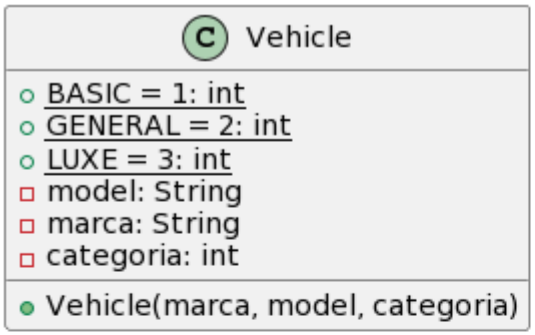
Peu informe:

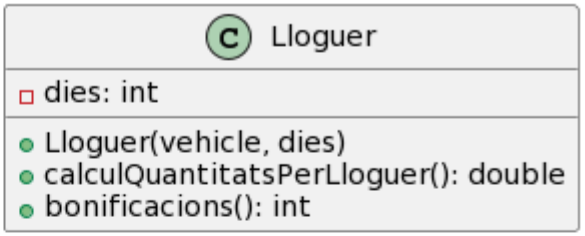
```
103     public String composaPeuHTML() {
104         Locale.setDefault(Locale.US);
105         String doubleFormat = String.format("%.1f", this.importTotal());
106         return String.format("<p>Import a pagar: <em>%s€</em></p>%n<p>Punts guanyats: <em>%d</em></p>", doubleFormat, this.bonificacionsTotal());
107     }
108 }
```

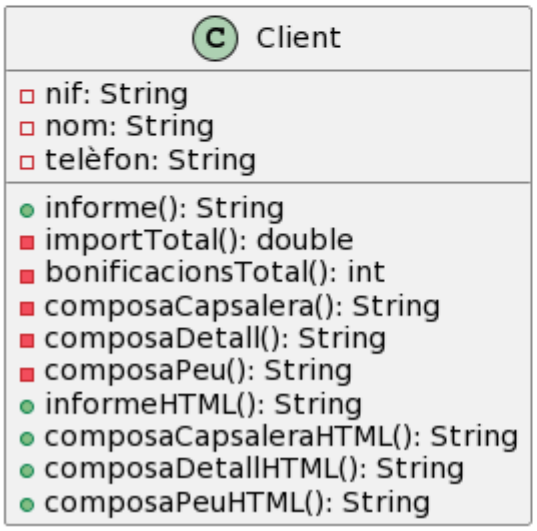
! Nota: he fet ús de la classe Locale per tal de canviar la regió del sistema per els mètodes que necessitaven reformatejar les dades de tipus double i em donaven problemes per separar els decimals amb un . en lloc de les comes. Continuem.

Aquest ha sigut el general el canvi final que li hem aplicat al nostre programa refactoritzat i per últim farem una ullada als diagrames de classes i de seqüència per veure la seva transformació final. Hem de tenir en compte que els anteriors diagrames de classes tenien inclós l'error del que hem parlat abans. Per tant, ara no trobarem el mètode "colat" dins del diagrama.

ACTUALITZACIÓ DIAGRAMA DE CLASSES:







CONCLUSIONS

Fins ara, hem pogut fer-nos una idea del que tractarà el programa i hem pogut recrear els diagrames corresponent a partir d'un petit escrit de codi.
El següent pas ja es centrarà en començar a construir el codi o fer les proves unitàries necessàries per començar, s'anirà documentant tot aquí.

Després de donar les primeres passes, ja tenim les que seran les bases del programa. Hem creat les 3 classes que teniem representades al *Diagrama de Clases* i posteriorment hem generat ni més ni menys que 16 proves unitàries que ens ajuden a comprovar que tot funciona correctament encara si fem alguna modificació com l'última que hem dut a terme.
Aquesta petita modificació consistia en fer una extracció de mètode a Client.informe(), la qual hem realitzat correctament i ara podem veure que el mètode informe() s'ha tornat una mica més llegible i menys aparatós.

El següent pas consistia en fer un moviment de mètode. En concret deplaçar , i transformar lleugerament, el mètode Client.quantitatPerLloguer() a Lloguer.quantitat(). Aquest pas ha fet que el mètode es torni més llegible encara a l'igual que la classe Client, la qual ara es veu més neta del que estava abans.

Tornant a documentar la pràctica. En els pasos anteriors hem pogut posar en pràctica una altra vegada les diferents parts del procés de refacció i aquesta vegada hem efectuat totes les passes vistes però per modificar el mètode en que es calculaven les bonificacions de cada client.
Després de netejar el codi amb aquestes pràctiques, hem passat al següent punt que es l'eliminació de variables temporals i amb això ens hem carregat totes, o quasi totes, les variables que quedaven al mètode Client.informe().
Això, evidentment, ha generat un canvi en el diagrama de classes una altre vegada i hem deixat constància d'això refent-lo. Com a punt molt positiu el mètode Client.informe() està quedant molt però que molt net.

L'última modificació ha deixat un codi impolut en el mètode Client.informe(). Ha canviat el diagrama de classes afegint molt més mètodes dels que teniem al principi però ja hem aconseguit un codi llegible, clar i, sobretot, fàcilment modificable, a part que bastante independent uns mètodes d'altres, el qual genera bastant aprofitament per part de cadascun.

El pas final d'aquest projecte ha sigut crear els mètodes que creen l'informe HTML del client a base de unes proves unitàries que nosaltres mateixos hem fet.

Ha sigut una bona experiència el conèixer amb pràctica real el procés de refactoring i veure com podem convertir un codi desastrós en un maco de veure i de llegir, encara que això comporti la creació de moltíssims més mètodes.
Personalment quan vam començar a conèixer els mètodes em va causar una mica de rebuig el fet d'haver de separar tant el codi i sentir que no ho tinc a la vista. Però a dia d'avui amb tot el que hem vist, sense dubte és necessari i és la millor manera de crear un codi llegible i fàcilment modificable.