
DIARI D'UN REFACTORING

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

INDEX

INTRODUCCIÓ.....	3
DIAGRAMES.....	4
Diagrama de classes.....	4
Diagrama de seqüència.....	4
EXTRACCIÓ DE MÈTODES.....	5
CONCLUSIONS.....	7

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:

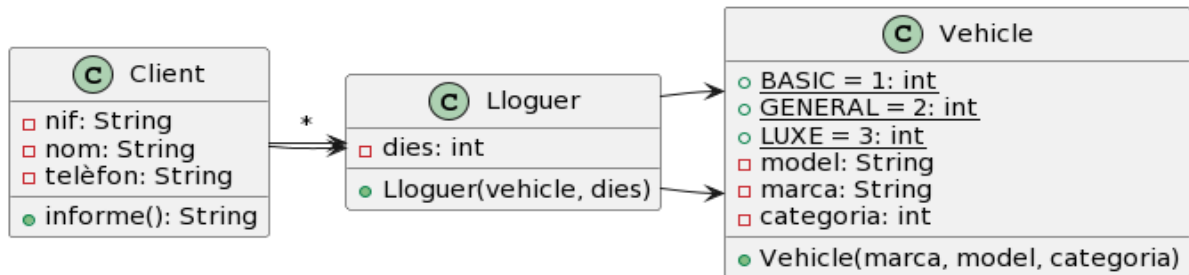
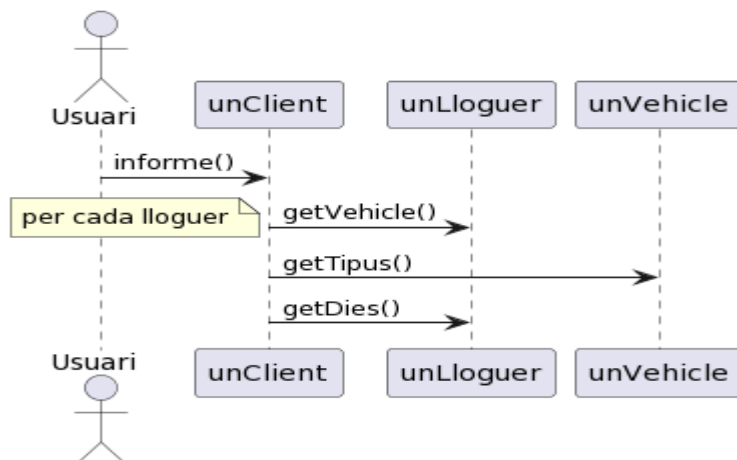


Diagrama de seqüència

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 ocorrides 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:

```
28
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 passat 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         // afegim lloguers freqüents
39         bonificacions ++;
40
41         // afegim bonificació per dos dies de lloguer de Luxe
42         if (lloguer.getVehicle().getCategoria() == Vehicle.LUXE &&
43             lloguer.getDies() > 1) {
44             bonificacions ++;
45         }
46
47         // componem 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     // afegim 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 }
```

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.