Disseny i Programació Orientats a Objectes

Memòria - LSGallos

Pol Piñol Castuera (pol.pinol)
Paula Fernández Lago (paula.fernandez)

Índex

1 Introducció

Aquest treball, LSGallos, és un projecte fictici creat per la universitat La Salle amb l'objectiu de gestionar una batalla de galls a Barcelona. Abans de començar amb el esdeveniment, shaurà de preparar i crear el programa informàtic per organizar d'una forma correcta i eficiente la competición. Aquesta part del treball es durà a terme per tots els estudiants de l'assignatura DPOO de la Salle.

El llenguatge escollit per la implementació del programa de gestió ha estat Java (versió 15.0.1) i el IDE utilitzat ha estat IntelliJ IDEA Ultimate 2020.2.3 x64. La raó per la qual hem escollit aquest IDE ha estat principalment per la facilitat que representa programar amb IntelliJ, ja que és un programa molt user-friendly el qual ajuda a l'usuari a corregir errors de codi i/o recomanacions de com optimitzar dit codi.

2 Resum de l'enunciat

La pràctica parteix de la idea que una empresa dedicada entre d'altres a organitzar batalles de gallos per diferents països demana un programa per organitzar i gestionar una única competició. Per el tant el projecte consisteix a simular una competició de batalla de galls.

La competició es configura amb el seu nom, les dates d'inici i final d'aquesta, i una llista amb els participants registrats a la competició fins al moment. De manera que el programa permet registrar nous participants en la competició fins que arriba la data d'inici.

La competició està constituïda per dues o tres fases celebrades en països diferents i cadascuna compte amb el seu pressupost. A la fase inicial participen tots els raperos que s'hagin registrat a la competició, a la intermitja , si n'hi ha , participen la meitat dels participants de la fase inicial que hagin obtingut major puntuació, i a la final només participen els dos millors raperos.

Cadascuna de les fases que formen una competició, alhora està formada per dues batalles, i cada batalla té assignat un conjunt de temes sobre els quals hauran de rapejar els participants. Les batalles poden ser de tres tipus diferents, escrita, les quals tenen un nombre de segons en el qual els participants han d'escriure les rimes, a sangre, on s'ha d'indicar el nom del productor de les bases, i per últim a capella. Per a cada batalla es generen aparellaments entre els participants de manera aleatòria. Se simulen totes les batalles exceptuant les de l'usuari que s'hauran de mostrar per pantalla.

A l'inscriure's a la competició cada rapero haurà d'indicar el seu nom, nom artístic, data de naixement, el seu nivell d'expertesa, un URL amb la seva foto i el seu país d'origen.

Per tal de desenvolupar el projecte es disposa de fitxers json amb la informació necessària sobre la competició i els raperos registrats en ella, i els temes sobres els que es pot rapejar amb les estrofes corresponents perquè puguin fer-les servir els raperos.

Pel que fa a l'execució del programa, el primer que farà serà carregar i tractar les dades del fitxer json pertinent i guardar-les a la RAM. A continuació es mostraran per pantalla les dades generals de la competició, tot seguit segons la data actual en relació amb la competició es mostrarà una de les opcions següents: si encara no hem arribat a la data d'inici de la competició, es mostrarà un menú que permeti a l'usuari registrar-se, per al qual es demanaran totes les seves dades i es comprovarà que siguin correctes, o sortir del programa, si per una altra banda ja ha començat la competició, però encara no ha acabat es mostrarà un menú com en el cas anterior, per fer el log-in, per al qual només haurà d'introduir el seu nom artístic. Per últim si la competició ja ha acabat, es mostrarà un missatge per pantalla que informarà sobre qui ha guanyat la competició. Un cop fets els aparellaments i simulades les batalles es mostrarà a l'usuari el lobby amb diferents opcions per escollir (batallar, mostrar ranquing, crea perfil o abandonar).

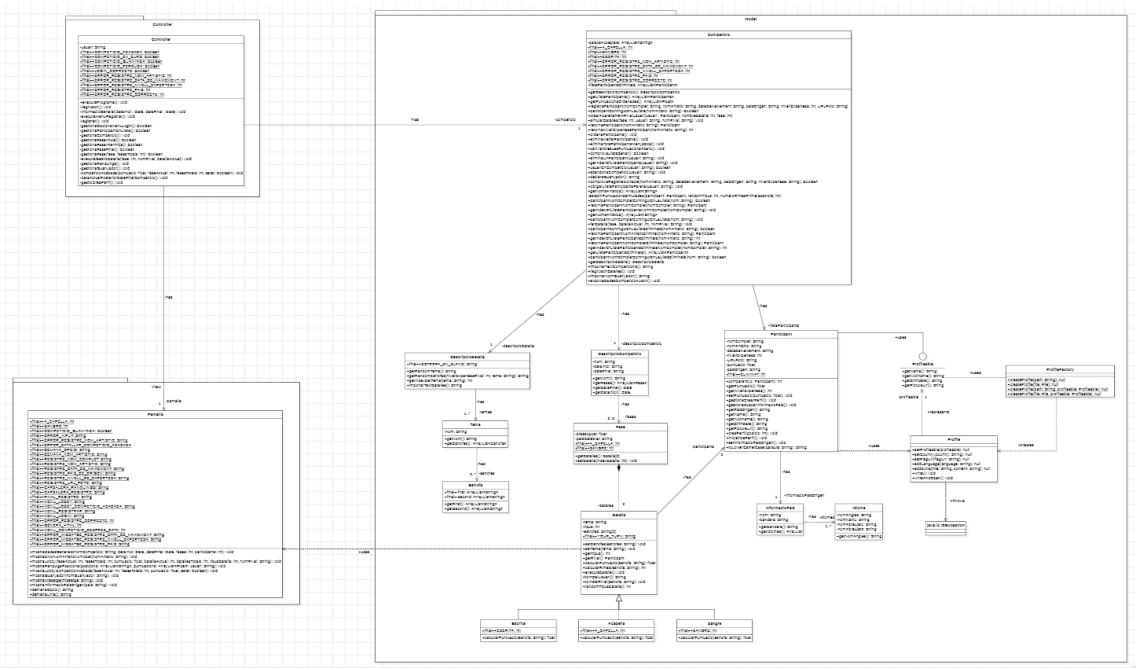
La batalla de l'usuari s'anirà mostrant progressivament per pantalla, de tal manera que primer sortirà el tema i es triarà aleatòriament qui comença, durant els dos torns de cada participant, l'usuari escriurà per pantalla les seves estrofes mentre que el contrincant les obtindrà de les estrofes contingudes en el json segons el tema i el seu nivell.

Un cop es finalitza la competició, és a dir, ja s'ha realitzat totes les batalles, es mostra el guanyador i s'indica si ha guanyat l'usuari. Igualment, si es surt de la competició es finalitzarà el programa i es designarà un guanyador.

Per tal de regular la competició, a cada fase es puntua als raperos per cada batalla tenint en compte les rimes que escrigui cada rapero en cada estrofa que faci i el tipus de batalla que està realitzant.

Per a l'opció de crear perfil, es demana el nom del participant del qual es vol crear un perfil, s'obté la informació sobre el seu país d'origen d'internet mitjançant un webservice i es genera un fitxer HTML que mostri la informació bàsica del rapero.

3 Diagrama de Classes UML



El diagrama ha estat dissenyat seguint l'estructura del patró model-view-controller (MVC), que és un estil d'arquitectura de software que separa les dades en tres blocs diferents, els de l'aplicació (model), els de la interfície d'usuari (view), i els de la lògica de control (controller). Amb l'objectiu d'aconseguir un diagrama l'ordenat, clar i amb un bon disseny.

Primerament, dins de la capa controller trobem la classe amb el mateix nom que s'encarrega d'actuar com a intermediari entre el model i la vista, gestionant el flux d'informació entre ells i les transformacions necessàries per adaptar les dades a cada bloc, i per el tant dirigir l'execució del programa, distribuint qui i quan realitza cada acció.

La intenció principal del nostre disseny ha sigut que el controller només estigués relacionat un cop amb el model i un cop amb la interfície de l'usuari. Això ho hem aconseguit a través d'anar fent millor al disseny a cada fase. A més, també volíem que la interacció de la terminal, és a dir, "printsï "scans"estiguessin situats només amb view. També hem fet un bon ús de la programació, intentant fer servir el màxim de constants (encara que això afecta el UML a l'existir molts atributs) per la reutilització de mètodes, en especial èmfasi a la classe Pantalla.

Fixem-nos ara en la capa model. Aquí podem trobar tota la part central i classes pròpies d'una competició de la batalla de galls. A la fase 1, al no saber com implementar un bon disseny un altre cop, vam decidir crear com a classe central la classe Competició i totes les classes serien les seves derivades. A més, no vam repartir responsabilitats. Finalment, podem concloure que hem arreglat, en la majoria dels casos, aquesta centralització i responsabilitat central a la classe Competició. Sobretot hem repartit la responsabilitat dels seus propis mètodes a Participant i Batalla.

Per tant el nostre model intenta repartir les responsabilitats a les seves classes, però en molts casos gestionem abans. Per posar un exemple, tenim un mètode a controller que es diu gestionaPerfil per després cridar a creaPerfil a participant.

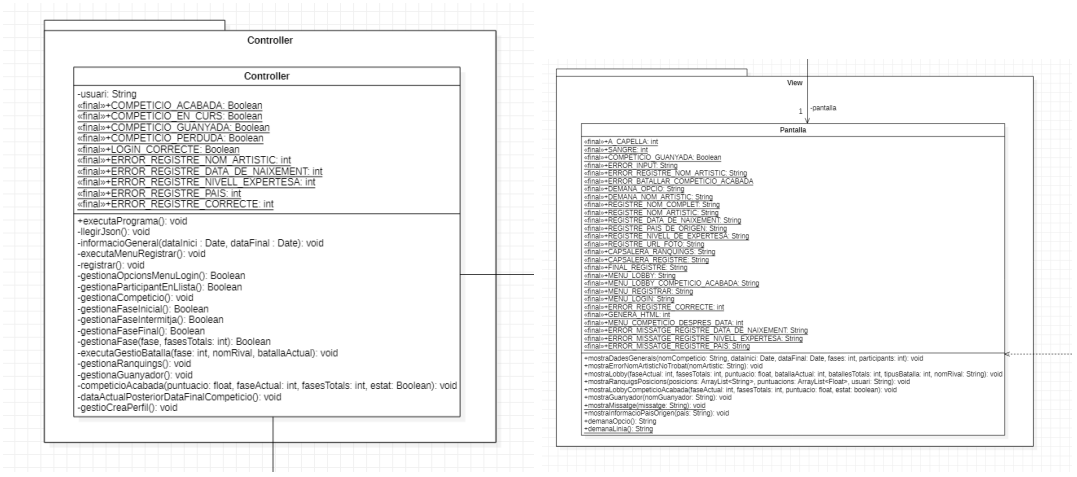
També s'hauria d'explicar l'existència de les classes DescripcioBatalla i DescripcioCompetició, que existeixen bàsicament per fer un bon l'ús d'escriptura i lectura del Json.

Caldria explicar en aquesta memòria un detall bastant important que no apareix en el UML referent a les batalles. Per simular una batalla, no s'han simulat aparellaments, sinó que només s'ha agafat l'usuari del programa i un rival aleatori i simulat les puntuacions dels altres participants (sempre vigilant que la llista quedés parella). Per simular aquestes puntuacions, no s'ha fet d'una forma aleatòria, sinó que s'ha simulat que cada participant simulés un nombre de rimes aleatòries entre 0,2,3 i 4 (donant més prioritat a rimes altes als participants amb més nivell d'expertesa) i es calculés la puntuació (amb les formules adients) de les dues suposades estrofes que faria amb aquelles rimes aleatòries. D'aquí es pot extreure dues conclusions: les puntuacions dels participants, tot i no utilitzar les estrofes del fitxer Json, treuen puntuacions igual de verídiques com si utilitzessin les estrofes del Json, i que la competició té un grau de dificultat alt, ja que quasi sempre a la final et sortirà un contrincant de nivell d'expertesa 2.

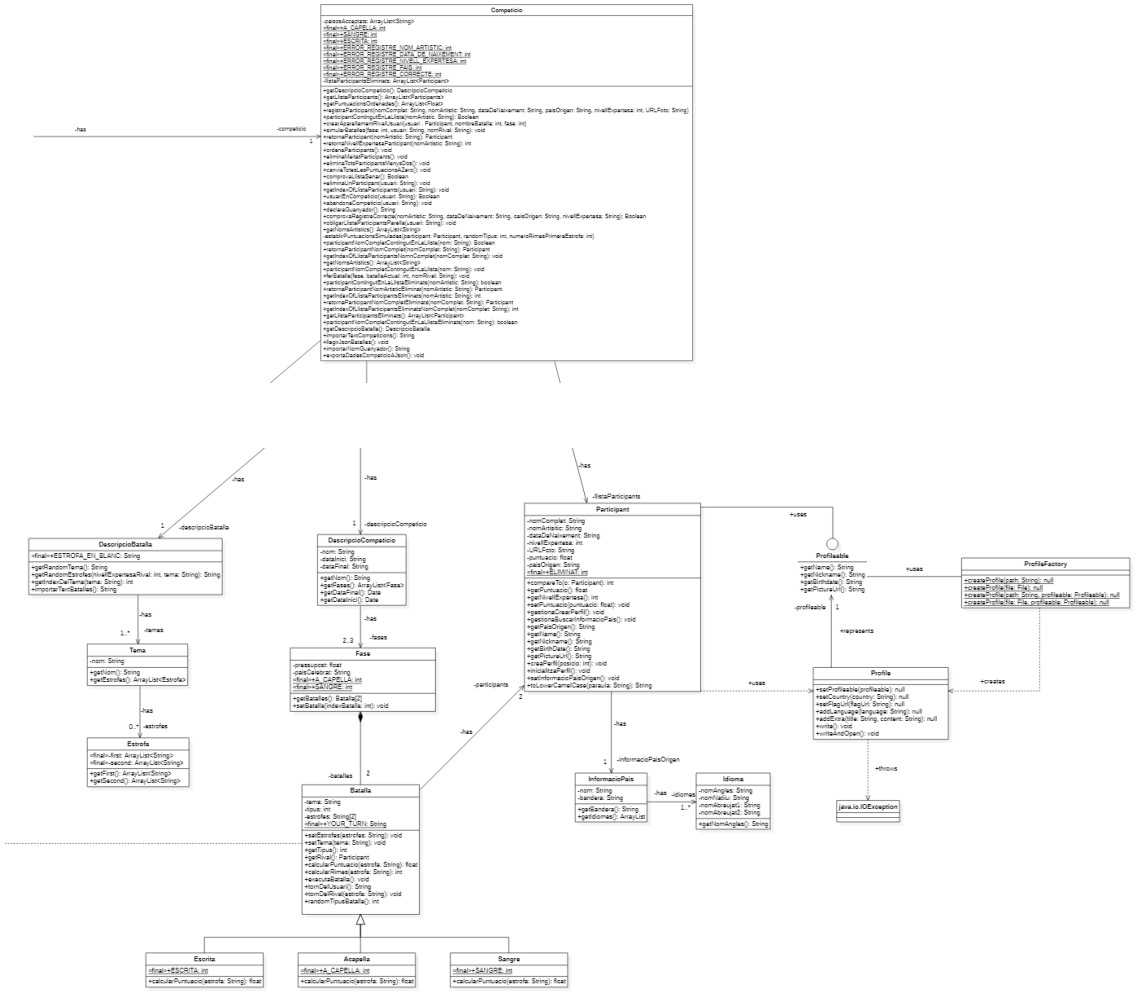
A més, es pot observar que Batalla és la superclasse de tres classes: Sangre, Acapella i Escrita. Aquesta herència està creada per fer un ús del polimorfisme amb el mètode calculaPuntuacio (utilitzem aquest mètode per calcular la puntuació d'una estrofa).

Finalment, hem de parlar de la classe Participant i les opcions creaPerfil. Aquestes implementacions estàn donades a la fase 4 del projecte, i no ens han suposat gaire esforç implementar-les però sí d'una forma eficient. Bàsicament hem fet que la classe participant implementa la interfície Profileable (amb el seu mètode abstracte) per fer ús dels mètodes de la classe Perfil i així generar un ftixer HTML. Si ens hi fixem, vam haver de crear dues classes més (País, atribut que abans era un String, i Idioma) per recollir de la forma que ens donava l'API anomenada REST COUNTRIES.

Com a conclusió d'aquesta secció, podem dir som conscients que hauríem de millorar la classe Participant, ja que en el nostre disseny està molt relacionada i depèn de moltes classes, i també que potser ens faltaria repartir algunes responsabilitats.



.


Competicio
-has
-competicio
DescripcioBatalla
-descripcioBatalla
DescripcioCompeticio
-descripcioCompeticio
Participant
-listaParticipants
+uses
Profileable
ProfilesFactory
Tema
-temes
Fase
-fases
Profile
-participants
-has
Batalla
-batalles
InformacioPais
-informacioPaisOrigen
Idioma
-idiomes
Estrofa
-estrofes
Escrita
Acapella
Sangre
java.io.IOException
-throws
+creates
-profileable
+represents

4 Mètode de proves

Per tal de provar el nostre codi i assegurar-nos del seu correcte funcionament hem fet servir diversos mètodes, però principalment hem anat provant el codi de fora manual canviant els fitxers amb les dades i provant diferents casos.

Per un costat, hem fet servir l'eina debugger per comprovar que les dades són correcte tant en l'API com en el fitxer json i també hem fet servir system.out.println per anar-nos assegurant del correcte funcionament de les diferents parts de la pràctica a mesura que les anàvem fent.

Hem provat el programa amb diferents fitxers json amb diferents competicions tant amb dos com amb tres fases per comprovar el funcionament dels dos casos. També hem provat de canviar les dades dels participants per comprovar que es detectessin els errors, hem registrat els nostres usuaris i provat el programa amb ells, hem fet els diferents tipus de batalles per comprovar que totes es realitzessin correctament. A més, hem provat de canviar al json la data per assegurar-nos del funcionament de totes les opcions en funció de si ha començat o acabat a competició.

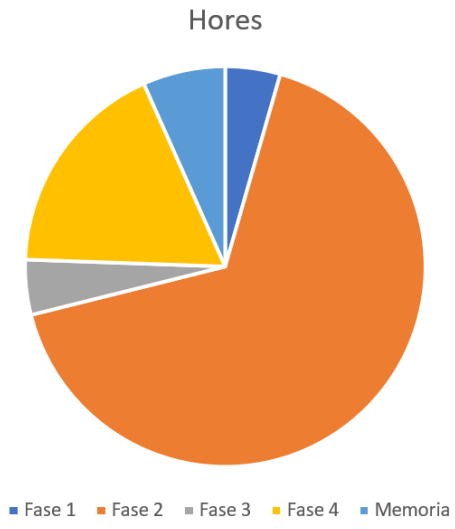
5 Dedicació en hores

Per a les fases 1 i 3 no li vam dedicar gaire temps inicialment, ja que només consistien a fer els diagrames UML i per tant era no hi havia darrere una feina més feixuga i pràctica com les altres fases, sinó més teòrica i d'aprenentatge per realitzar un bon disseny. Però, en rebre feedback sobre la fase 1 i la fase 3, de la mateixa forma en els dos casos, hem hagut de refer-les com que no hem sabut implementar un bon disseny de primera instància.

Per l'altra banda, a la fase 2 va ser la fase la qual vam haver de dedicar més hores en diferència, ja que havíem de fer tot el codi del programa en Java per a la seva correcta execució, investigar el funcionament dels fitxers json, per tal de poder llegir la informació, entre altres coses.

Finalment per a la fase 4, en general, ha sigut una conclusió de la pràctica al només haver d'utilitzar una llibreria donada i crear un perfil reutilitzant tot el codi programat a la fase 2.

Per concloure, podem dir que la distribució de temps dedicat a cada fase ha sigut la que es representa al següent gràfic.



6 Conclusions

El procés de programar i pensar durant les quatre fases del projecte ens ha fet entendre les bases d'un disseny i la programació orientada a objectes. Hem vist que, mitjançant classes i objectes que interactuen entre si, enviant-se missatges, i mitjançant classes Controller, que donen ordres al programa, es programa d'una forma molt estructurada.

En fer la pràctica en Java, i utilitzant programació orientada a objectes observem que tractem al programa com a un conjunt d'objectes que s'ajuden entre ells per realitzar accions i d'aquesta manera, aconseguim programes més fàcils d'escriure, entendre, mantenir i reutilitzar, permetent-nos programar de manera més productiva.

Com a conclusions tècniques, cal explicar que hem tigut problemes durant las fases de disseny en UML, on ens hem trencat més el cap, encara que a mesura que hem anat fent la part més pràctica en Java hem acabat d'assolir els coneixements i finalment ho hem entès tot. La part del disseny UML, en específic, la primera presa de contacte a les dues fases que s'ocupaven d'això, no havíem acabat d'entendre exactament el que havíem de fer. Encara que, després de rebre el feedback en aquestes fases, les hem assolit correctament.

Com a conclusions més personals, cal dir que ens ha agradat molt el projecte encara que l'hem trobat bastant llarg; entenem que és una única pràctica i a més, aquesta assignatura no té exàmens, però això no impedeix que ha sigut el projecte més llarg que hem fet durant el grau. Gràcies a la llargada del projecte, hem après moltes coses sobre el disseny orientat a objectes i en específic, sobre la programació en Java, permetent-nos programar de manera més eficaç i habitual en aquest llenguatge de programació.

Referències

[1] *MVC Design Pattern. Geeks for geeks*, 2018 [consulta 21-12-2020]. Disponible en: `https://www.geeksforgeeks.org/mvc-design-pattern/`

[2] *A short overview of software design with GRASP.* Tonny Garic, 2018 [consulta 16-11-2020]. Disponible en: `https://tonnygaric.com/blog/a-short-overview-of-object-design-with-grasp`

[3] *The Java Tutorials. Oracle Java Documentation*, 2020 [consulta 15-11-2020]. Disponible en: How to Throw Exceptions (The Java™ Tutorials > Essential Classes > Exceptions) (`oracle.com`)

[4] *UML class diagram tutorial.* Visual Paradigm [consulta 5-10-2020]. Disponible en: UML Class Diagram Tutorial (`visual-paradigm.com`)

[5] *6 OOP Concepts in Java.* Raygun, 2018 [consulta 14-10-2020]. Disponible en: 6 OOP Concepts in Java with examples · Raygun Blog