UNIVERSITAT DE BARCELONA

Pràctica 4 Disseny de Software

Márquez Vara, Noah
20392536

Aresté Saló, Lluc
20460440

28 Desembre 2021

ÍNDEX

1 Model de Domini

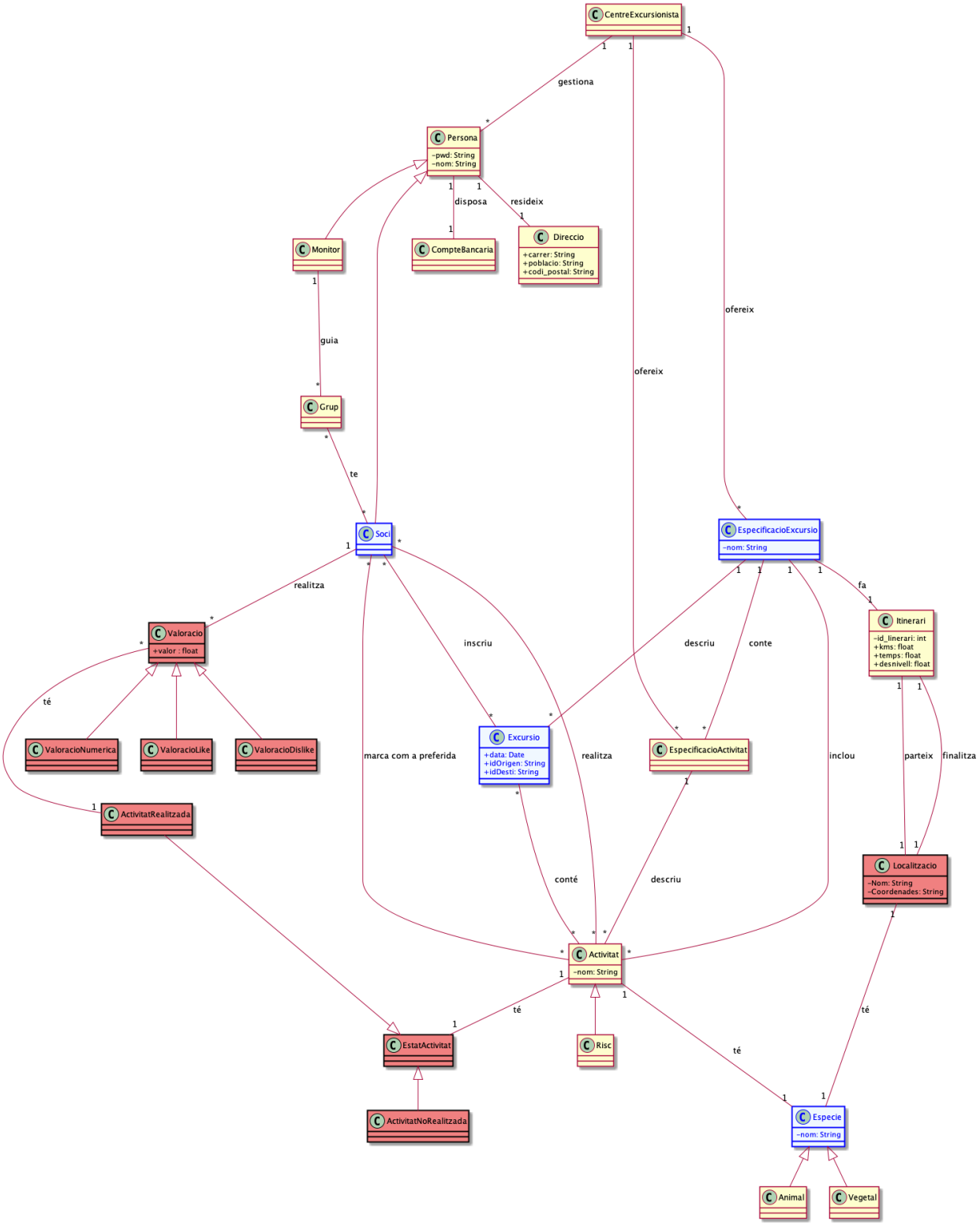

Figure 1.1: Model de Domini

Respecte a la pràctica 1 s'han afegit els diferents tipus de valoracions i la Localització, a més de que les excursions tenen ara un origen i un destí a una localització.

2 MILLORES & CANVIS PRÀCTICA 3

Repecte a la pràctica 3 hem fet bastantes millores, unes més significatives que d'altres. A continuació farem un llistat amb totes les millores i optimitzacions realitzades amb la corresponent explicació:

1. El canvi que s'ha d'esmentar primerament és el que hem fet en relació a solucionar unes crides errònies que feiem a la *Pràctica 3*, ja que feiem crides als subcontroladors des del *model*, i això no és correcte. Mostrem com a exemple d'aquest error uns diagrames de flux per comparar l'execució del mètode ***afegirActivitatRealitzada***:

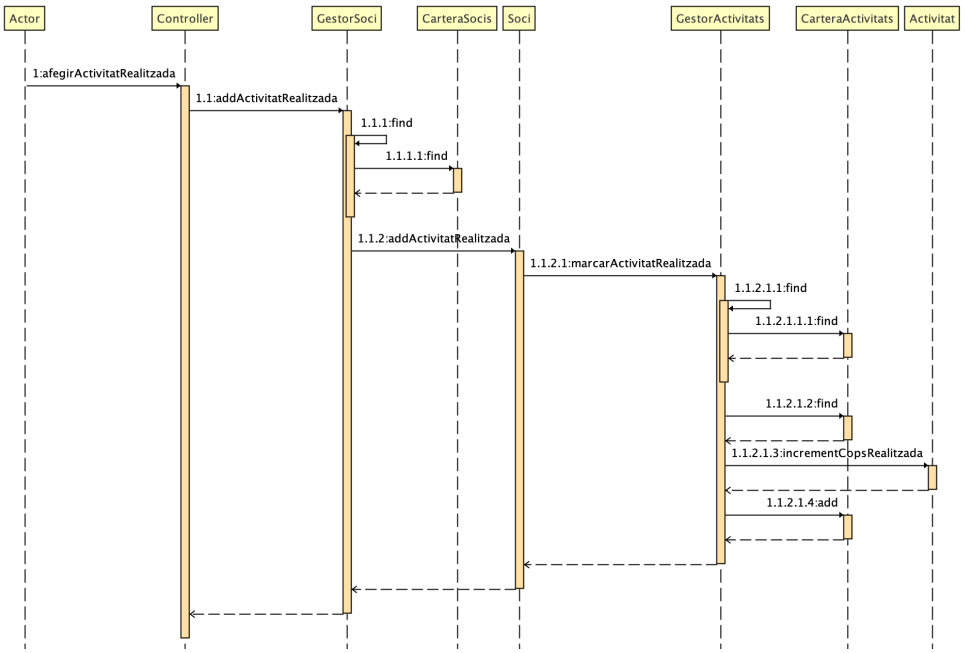

Figure 2.1: afegirActivitatRealitzada (before)

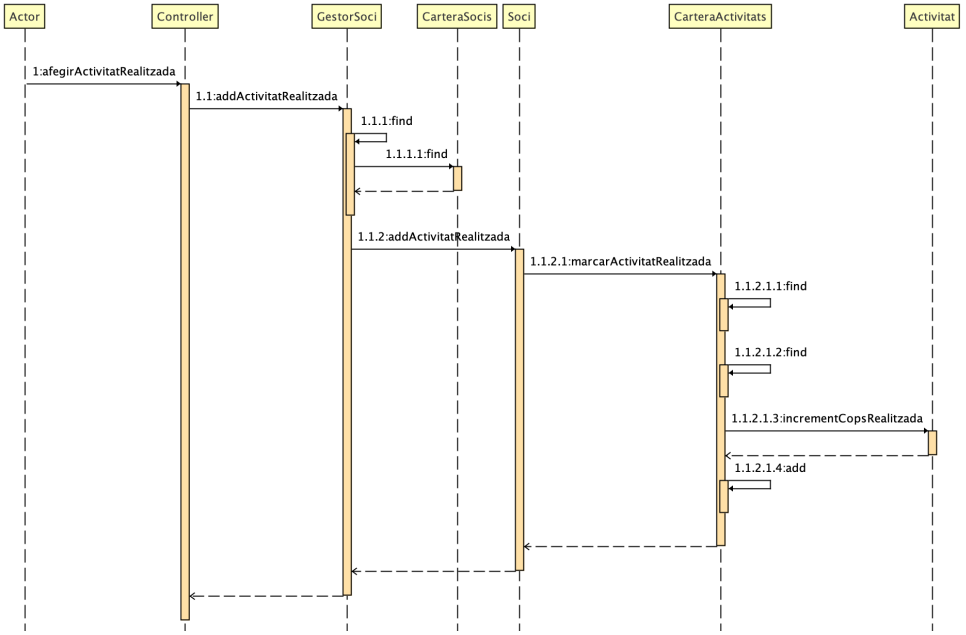

Figure 2.2: afegirActivitatRealitzada (after)

Podem apreciar com ara el **Soci** (model) no torna a cridar a un *controller* (*GestorActivitats*), sinó que directament es relaciona amb un altre classe del model: **CarteraActivitats**.

2. La nostra classe **Controller** actua com a *Façana* de la **Vista** respecte als altres controladors i el **Model**. És a dir, ens proporciona una interfíce unificada a un conjunt d'utilitats d'un subsistema (el model) complex. Ens redueix la corba d'aprenentatge necessària per aprofitar i entendre el subsistema esmentat anteriorment.
3. Hem reduït de manera notòria la quantitat de mètodes que tenia la classe **Controller** unificant-la molt més i reutilitzant codi per tal d'evitar repeticions innecesàries. A més hem aprofitat per desfer-nos de mètodes que no s'utilitzaven, bé perquè no ens feien falta per la nostra implementació o bé perquè ho realitzavem d'un altre manera que creiem més convenient.
4. En general hem desacoblat i cohesionat més els nostres gestors (subcontroladors), ja que abans realitzaven l'execució d'alguns mètodes íntegrament, i això ho hem traspassat a les corresponents carteres (al model), per tal de delegar la responsabilitat a les classes que es consideren expertes en una certa informació (i.e. *Espècies, Activitats, Socis, ...*). A continuació esmentem els canvis realitzats a cada gestor:

    a) **GestorActivitats:** Com hem esmentat anteriorment, els gestors realitzaven alguns mètodes i aquests era millor situar-los en les seves classes expertes. En aquest cas, *GestorActivitats* realitzava els mètodes de ***visualitzaActivitat*** (per donar l'informació d'una activitat), ***valorarActivitat*** (per valorar una activitat i fabricar l'*String* que es retornaria), ***marcarActivitatRealitzada*** (tal i com indica el nom, per marcar com a realitzada una activitat i buscar-la en el sistema) i per últim ***marcarActivitatPreferida*** (per a marcar una activitat com a preferida i buscar-la en el sistema). Tots aquests mètodes els realitza ara la classe **CarteraActivitats**, que és l'experta en les activitats. A més, hem eliminat alguns mètodes que ja no feiem servir, i hem juntat d'altres per tal d'evitar la repetició de codi.

    b) **GestorBaseDaddes**: En el cas d'aquest gestor no hem fet gairebé cap canvi, només hem eliminat un mètode que ja no el feiem servir en aquesta classe: ***findSoci***.

    c) **GestorEspecies**: Per aquest gestor només hem eliminat tres mètodes que ja no feiem servir en la nostra implementació, o que es realitzen d'un altre forma a un altre classe.

    d) **GestorExcursions**: En aquest gestor si que hem fet més canvis, com per exemple eliminar alguns mèotodes que ja no es feien servir. A més, igual que ha succeït amb la classe **GestorActivitats** anteriorment, també hem delegat molts dels seus mètodes a les classes expertes (e.g. *CarteraExcursions*) per tal de rebaixar la seva responsabilitat.

    e) **GestorLocalitzacions**: Per aquesta nova pràctica hem creat aquest nou gestor, per tal de gestionar les localitzacions de les excursions i de les espècies. Gestiona pocs mètodes en comparació amb els altres gestors; entre aquests es troben obtenir la pàgina web d'aquesta, obtenir les espècies d'una localització i per últim, llistar el catàleg de localitzacions del sistema.

    f) **GestorSoci**: Aquest gestor no ha variat gaire, només s'han mogut alguns mètodes per tal de no realitzar la implementació en el gestor, sinó en altres classes (e.g. **CarteraSocis, CarteraActivitats**). Un canvi interessant és que ara a l'hora de validar el registre d'un soci, s'inicializen les carteres d'aquest i s'afegeix el soci a la cartera de socis generals, guardada a la classe ***CeXtrem***, per tal d'assegurar el correcte funcionament de l'aplicació.

5. **ValidarDadesSoci**: Ara la present classe l'hem mogut a la carpeta dels controladors ja que considerem que és on hauria de ser-hi, perquè *gestiona* les dades del soci a l'hora de fer un registre. A més, hem eliminat alguns mètodes que eren innecesaris per la nostra

implementació.

6. **Activitat**: Hem afegit un parell d'atributs: **copsRealitzada & idExc**. Aquests atributs, com el propi nom indica, seran per portar constància dels cops que s'ha realitzat una certa activitat i a quina excursió està relacionada. Per aquest motiu s'han afegit els corresponents *getters & setters*, un mètode per incrementar en 1 els cops que s'ha realitzat l'activitat i per últim, un mètode per comparar dues activitats pels cops que s'ha realitzat cadascuna (ens serà útil a l'hora de fer el *Top 10*).
7. **CarteraActivitats**: Hem reutilitzat molts mètodes per tal d'evitar replicar codi de manera innecessària. Ara la cartera realitza tots els mètodes relacionats amb activitats, ja que és l'experta en la seva informació. Entre aquests mètodes es troben: ***getActivitatsOrdenadesAlfabeticament*** (útil a l'hora de llistar activitats preferides, realitzades o pertanyents a alguna excursió en ordre alfabètic), ***getTop10Valorades*** (ens retorna el top10 d'activitats valorades segons un tipus de valoració: numèrica, like o dislike), ***getTop10Realitzades*** (com el propi nom indica, ens retorna el top10 d'activitats realitzades en la nostra aplicació), ***marcarActivitatRealitzada*** (aquest mètode comprova que el soci no hagi realitzat anteriorment l'activitat i l'afegeix a la seva cartera d'activitats realitzades), ***valorarActivitat*** (aquest mètode comprova que el soci hagi realitzat anteriorment l'activitat, que no l'hagi valorat anteriorment i posteriorment procedeix a valorar-la segons el tipus de valoració desitjat), ***getInfoActivitat*** (aquest mètode ens retorna la informació de les valoracions d'una activitat, és a dir, el número de likes/dislikes i la valoració mitja), ***getNomExc*** (ens retorna l'excursió a la que està associada l'activitat) i per acabar ***calcValoracio*** (ens calcula la valoració de l'activitat segons el tipus desitjat).
   Ara a més, aquesta classe fa un *extends* d'*Observable*, però això s'explicarà detalladament en l'apartat 4. Patrons utilitzats a la Pràctica 4.
8. **Especie**: L'únic canvi respecte la pràctica 3 és que ara ja no guarda l'ID d'una excursió, sinó el d'una localització (ja que ara les espècies són autòctones d'una localització i així es relacionen amb les diferents excursions). Tampoc guarda el seu ID, ja que aquest el feiem servir a la pràctica 3 per tenir vàries unitats d'una espècie. En l'actual implementació ja no ens fa falta.
9. **CarteraEspecies**: A banda d'eliminar alguns mètodes que ja no s'utilitzaven, també s'ha afegit un mètode que ens llista el catàleg d'espècies disponibles en el nostre sistema.
10. **Excursio**: Ara una excursió ja no guarda una cartera d'espècies, ja que les relacionem mitjançant una localització. A més, guarda dues localitzacions, una d'origen i una de destí. A més, hem aprofitat per desfer-nos d'alguns mètodes que no ens eren útils per la nostra implementació.
11. **CarteraExcursions**: Hem afegit un mètode per llistar les espècies que són presents en una excursió, relacionat amb la localització. És a dir, com les excursions tenen una localització d'origen i una de destí, i les espècies són autòctones d'una localització, aprofitem aquesta relació per veure quines espècies seràn vistes en cada excursió.
    Té un mètode per a poder llistar el catàleg d'excursions de la pràctica pel seu nom.
    Hi ha també un mètode per a llistar les excursions ordenades per la data de realització d'aquestes.
    I per últim, hi ha un mètode per llistar les excursions disponibles en una localització.
12. **Localitzacio**: Nova classe d'aquesta pràctica. Representa diferents localitzacions de la nostra pràctica. Té un nom, una pàgina web i una cartera d'espècies per tal de guardar les espècies autòctones d'una certa localització.
13. **CarteraLocalitzacions**: Nova classe pràctica 4. Ens gestionarà les funcionalitats relacionades amb les nostres localitzacions i farà funcions similars a les altres carteres, a més de llistar el catàleg de localitats de la nostra aplicació i les espècies d'una de les localitzacions.

14. **Soci**: Ara el soci també guarda una cartera d'activitats valorades, per evitar que pugui valorar més d'un cop una activitat. A més, també afegeix els observadors a la seva cartera d'activitats realitzades (explicació detallada a l'apartat 4. Patrons utilitzats a la Pràctica 4).
    I com hem comentat anteriorment, ara el soci no demana dades a cap controlador, només es connecta amb classes del model.
15. **CarteraSocis**: Cap millora o canvi.
16. **Valoracio**: Ara la classe valoració guarda també un ID de l'activitat a la que està associada, per tal de poder inicialitzar les dades de prova a l'inici de l'aplicació. Té un constructor per afegir els tipus de valoracions del sistema a un *HashMap* i un altre constructor per iniciar algunes activitats amb valoracions a l'inici de l'aplicació. La resta de mètodes continua igual, l'única diferència respecte a la pràctica 3, és que ara no tenim una classe ***ValooracioLikeDisLike***, sinó que ho hem separat en dues, per fer-ho més adient com es veurà a continuació.
17. **ValoracioDislike**: "Nova" classe en aquesta pràctica, ja que hem separat Likes i Dislikes en dues classes diferents. A la pràctica 3 implementava el patró de *Singleton* però hem vist que no ens era necessari. Ara aquesta classe tracta els dislikes, i a més hem eliminat el mètode que comprovava si la valoració era correcta, ja que ara la nostra aplicació funciona amb una interfície gràfica, i hem restringit la forma en què el soci valora, per tal de que no es pugui equivocar. El mètode ***getInfo*** (que retornava la informació d'aquest tipus de valoració) ha canviat la forma en retornar l'*String* per fer-ho més adient per mostrar per pantalla a l'hora de mostar l'*EscenaActivitat*.
18. **ValoracioLike**: Mateixa explicació que l'esmentada en **ValoracioDislike**.
19. **ValoracioNumerica**: A la pràctica 3 implementava el patró de *Singleton* però hem vist que no ens era necessari. Hem eliminat el mètode que comprovaba si la valoració era correcta, ja que ara per valorar numèricament s'ha de fer ús d'un *slider* que comprèn els valors de l'1 al 5. A més hem modificat la forma en què el mètode ***getInfo*** mostra la valoració mitjana, per tal d'adaptar-ho a la interfície gràfica.
20. **ValoracioStrategy**: Només s'ha eliminat el mètode per comprovar si una valoració era correcta.
21. **CeXtrem**: Ara a més de les carteres que ja guardava a la pràctica 3 (per guardar la informació general de l'aplicació), també guarda una cartera de localitzacions.
    El seu constructor ja no inicialitza les carteres, perquè ens donava problemes a l'hora d'aplicar el patró *Observer* i a més perquè no tenia gaire sentit inicialitzar-les al constructor, quan feiem un *set* de les dades que rebiem del *DataService* (base de dades).
22. **DAOActivitatMOCK**: Només s'ha afegit una activitat més per tal de testejar correctament els tests i el funcionament del sistema.
23. **DAOActivitatsRealitzadesMOCK**: Nou *DAOMOCK* que s'ha afegit per aquesta pràctica, per tal d'inicialitzar activitats realitzades als socis, i així evitar fer-ho des dels tests. A més, implementa dos nous mètodes per tal d'obtenir les dades que s'inicialitzen d'una manera diferent als altres *DAOMOCK*, per tal de poder relacionar els socis amb aquestes activitats realitzades a la classe del ***DataService***.
24. **DAOEspecieMOCK**: S'han afegit molts més exemples d'espècies per tal de tenir almenys una espècie per localització. A més, s'han eliminat els ID's de les espècies, ja que en aquesta pràctica ja no els fem servir. I ara les relacionem amb ID d'una localitzacó en lloc d'un ID d'una excursió.
25. **DAOExcursioMOCK**: Ara a més del nom i la data de l'excursió, també inicialitza el seu origen i el seu destí.
26. **DAOLocalitzacioMOCK**: Nou *DAOMOCK* que inicialitza les localitzacions de la nostra aplicació amb el seu nom i la seva pàgina web.

Implementa els mateixos mètodes que els altres *DAOMOCK*.

27. **DAOSociMOCK**: Cap canvi/millora respecte la pràctica 3.
28. **DAOValoracioMOCK**: Nou *DAOMOCK* que inicialitza diferents valoracions a algunes activitats per tal d'inicialitzar algunes dades de prova per l'aplicació i sobretot pels tests, ja que ara no inicialitzem cap dada des d'ells.
    Implementa els mateixos mètodes que els altres *DAOMOCK*.
29. **DAOS** (interfícies): Cap canvi/millora respecte la pràctica 3, llevat de que s'han afegit tres noves interfícies per treballar amb els tres nous *DAOMOCK* esmentats anteriorment. A més, la interfície ***DAOActivitatsRealitzades*** inclou dos nous mètodes per tal d'obtenir les dades del *HashMap* de ***DAOActivitatsRealitzadesMOCK***. Aquests mètodes ens permeten obtenir les *keys* del *HashMap* i els *values* en forma de *Collection<List<String»*, per tal de tractar aquestes dades a la classe del ***DataService***.
30. **AbstractFactoryData**: Hem afegit els tres *DAOMOCK* nous.
31. **DataService**: S'han afegit com atributs els nous *DAOMOCK* i s'inicialitzen en el constructor de la classe fent ús de l'***AbstractFactoryData***.
    S'han afegit varis mètodes per relacioonar les espècies amb les localitzacions, les excursions amb les localitzacions, les activitats amb les excursions, les activitats amb les valoracions i els socis amb algunes activitats realitzades. Tot això per evitar guardar atributs no POJO (*Plain Old Java Object*) a les bases de dades.
    Ara també s'inicialitza la cartera de localitzacions al mètode ***inicialitzaCarteres***.
32. **FactoryMOCK**: S'han afegit les *FactoryMOCK* necessàries pels nous *DAOMOCK*.

A moltes classes feiem comprovacions del tipus:

1. Existeix el soci?
2. Existeix l'activitat?
3. Existeix l'excursió?

Però ara a l'implementar la pràctica en una interfície gràfica, aquesta ens presentarà les dades que té el sistema, i no farà falta comprovar si una classe de les esmentades anteriorment existeix o no. Això ens redueix les històries d'usuari a pensar (l'*Excel* estarà a la carpeta *docs* de la pràctica) i també ens redueix el número de comprovacions que hem de realitzar als mètodes de totes les classes.

Un altre canvi notable és el fet de no inicialitzar dades de prova en els tests. Com s'ha mencionat anteriorment, s'han creat 2 nous *DAOMOCK* per inicialitzar les dades de prova de la nostra aplicació per tal de comprovar el funcionament dels tests i veure algunes dades a l'iniciar l'aplicació.

Com a últim canvi, és el fet obvi de que ara la nostra aplicació funciona en una interfície gràfica. Aquest fet no ens ha fet canviar gairebé res del nostre codi, ja que com el teniem prou bé, només hem hagut d'enllaçar les coses necessàries.

Ara la nostra classe ***AppMain*** és la que ens inicialitza l'aplicació, fent un *Controller.getInstance()* per tal d'evitar que tingui la responsabilitat de crear un nou controlador, així ens aprofitem dels avantatges d'haver realitzat la classe ***Controller*** fent ús del patró *Singleton*. El nostre controlador llavors serà l'encarregat de i fer de pont de comunicació entre la vista (a la que l'indiquem qui és el nostre controlador) i el model.

3 PATRONS UTILITZATS A LA PRÀCTICA 4

Com a la pràctica 3 ja vam aplicar bastants patrons i tots de manera bastant correcte, en aquesta pràctica hem considerat que només ens calia implementar el patró *Observer* per tal d'aconseguir la funcionalitat que se'ns demanava a l'enunciat. Aquesta funcionalitat consistia en què, un cop implementades les llistes del Top10 activitats fetes/realitzades i del Top10 activitats valorades numèricament, si es realitzava/valorava una activitat, aquestes llistes s'havien d'actualitzar automàticament.

Aquesta funcionalitat s'aconsegueix aplicant el patró *Observer*, de manera que si es realitza o es valora una activitat, la vista s'actualitzarà automàticament. Hem codificat el patró de tal forma que només s'actualitzi la llista que hagi patit un canvi, és a dir, si només s'ha valorat una activitat només actualitzarem el Top10 d'activitats valorades, mentre que el Top10 d'activitats realitzades no s'actualitzarà.

El patró *Observer* és un patró de disseny de comportament que permet definir un mecanisme de subscripció per notificar diversos objectes sobre qualsevol esdeveniment que passi a l'objecte que estan observant. Nosaltres l'utilitzem ja que els canvis d'estat en les valoracions d'una activitat o si aquesta s'ha marcat com a realitzada, requereixen fer canvis en l'***EscenaMain***.

Hem implementat el patró fent ús de l'interfície *Observer* a la nostra classe ***EscenaMain*** i fent un *extends* d'*Observable* a la nostra classe ***CarteraActivitats***. D'aquesta manera, quant la cartera d'activitats valora o marca com a realitzada una activitat, crida al mètode ***setChanged*** per indicar que ha canviat i posteriorment crida al mètode ***notifyObservers*** per notificar als seus observadors (en aquest cas ***EscenaMain***).

A continuació mostrem els esquemes de com hem aplicat el patró:

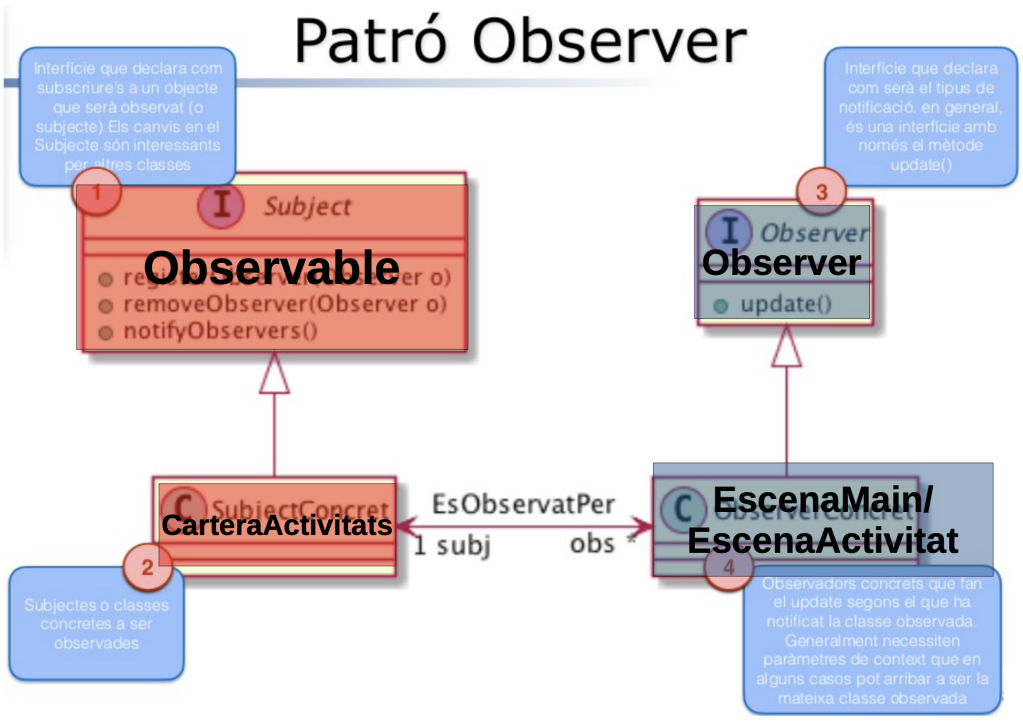

Figure 3.1: Esquema *Observer Pattern*

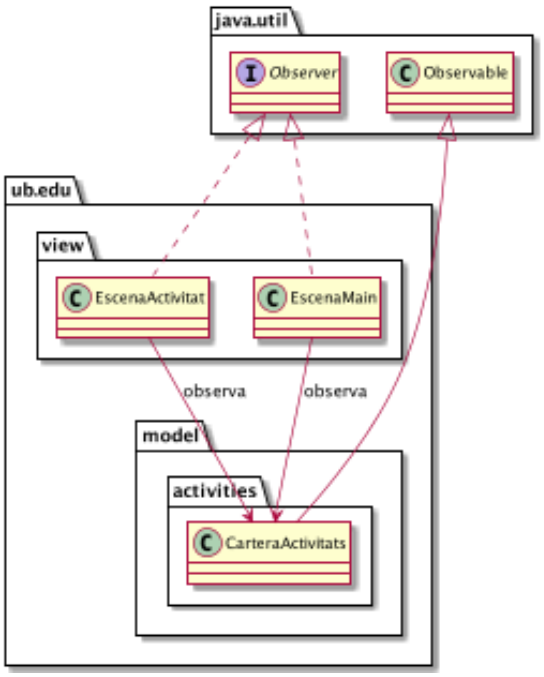

Figure 3.2: Esquema *Observer a l'aplicació*

Com es pot comprovar, a més de la classe ***EscenaMain***, també implementa el patró *Observer* la classe ***EscenaActivitat***. Vam considerar adient que com aquesta escena mostra la informació d'una activitat en concret, i entre aquesta informació es troben les seves valoracions (i.e. likes, dislikes, valoració mitja), seria interessant que s'actualitzés la informació si valoravem una activitat, i així ho hem fet.

4 Diagrama de Classes

Adjuntarem imatges de tots els diagrames de classes de la nostra pràctica, alguns més detallats i altres amb una visió més generals. Tots aquests diagrames estaran guardats a la carpeta *docs* per tal de poder visualitzar-los amb més detall.

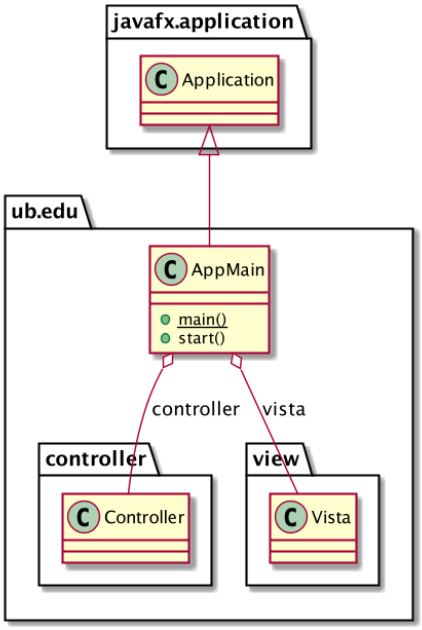

Figure 4.1: Iniciació interfície gràfica

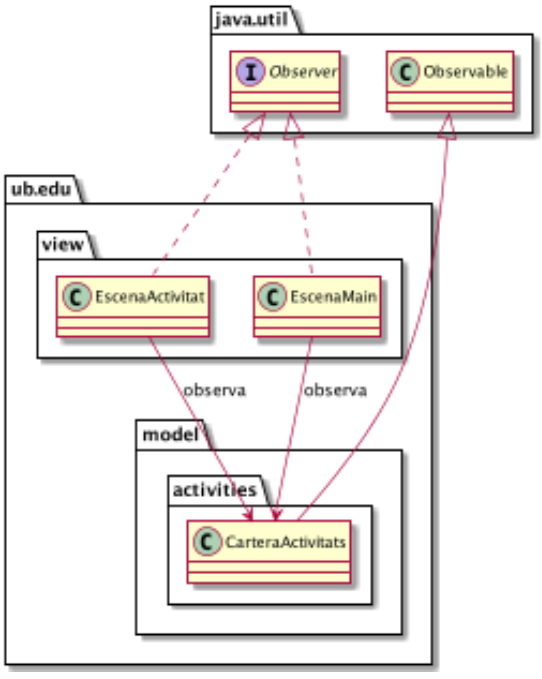

Figure 4.2: Patró Observer

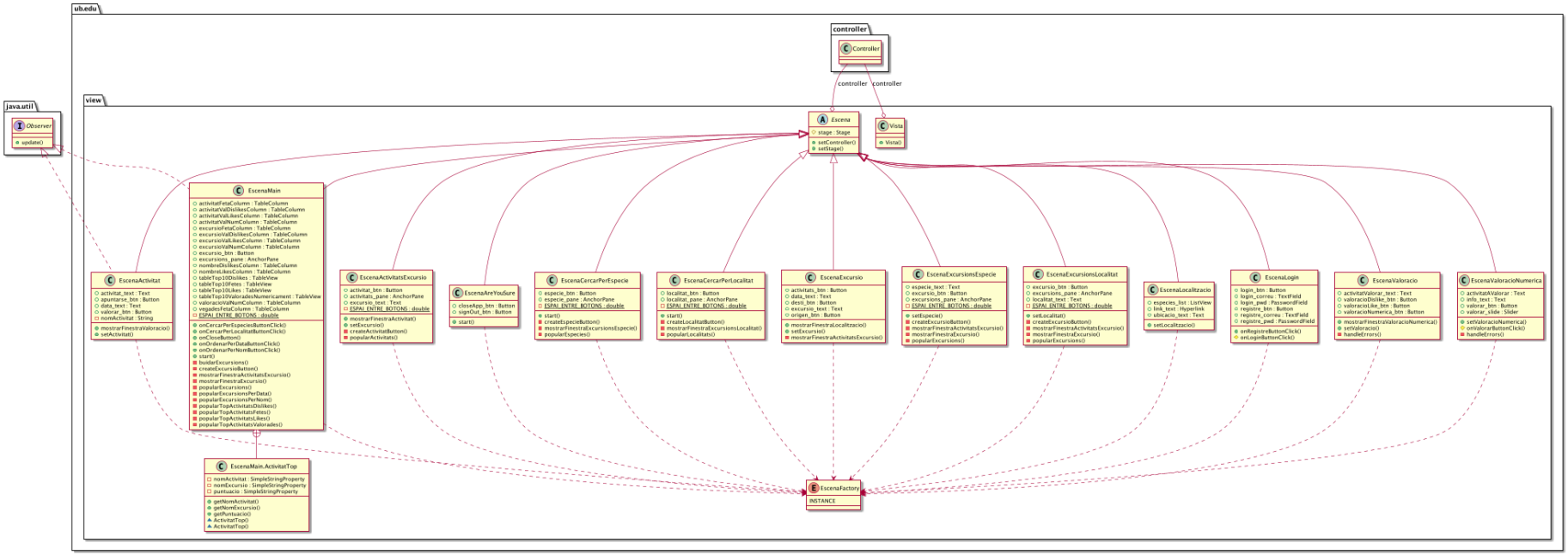

Figure 4.3: Vista

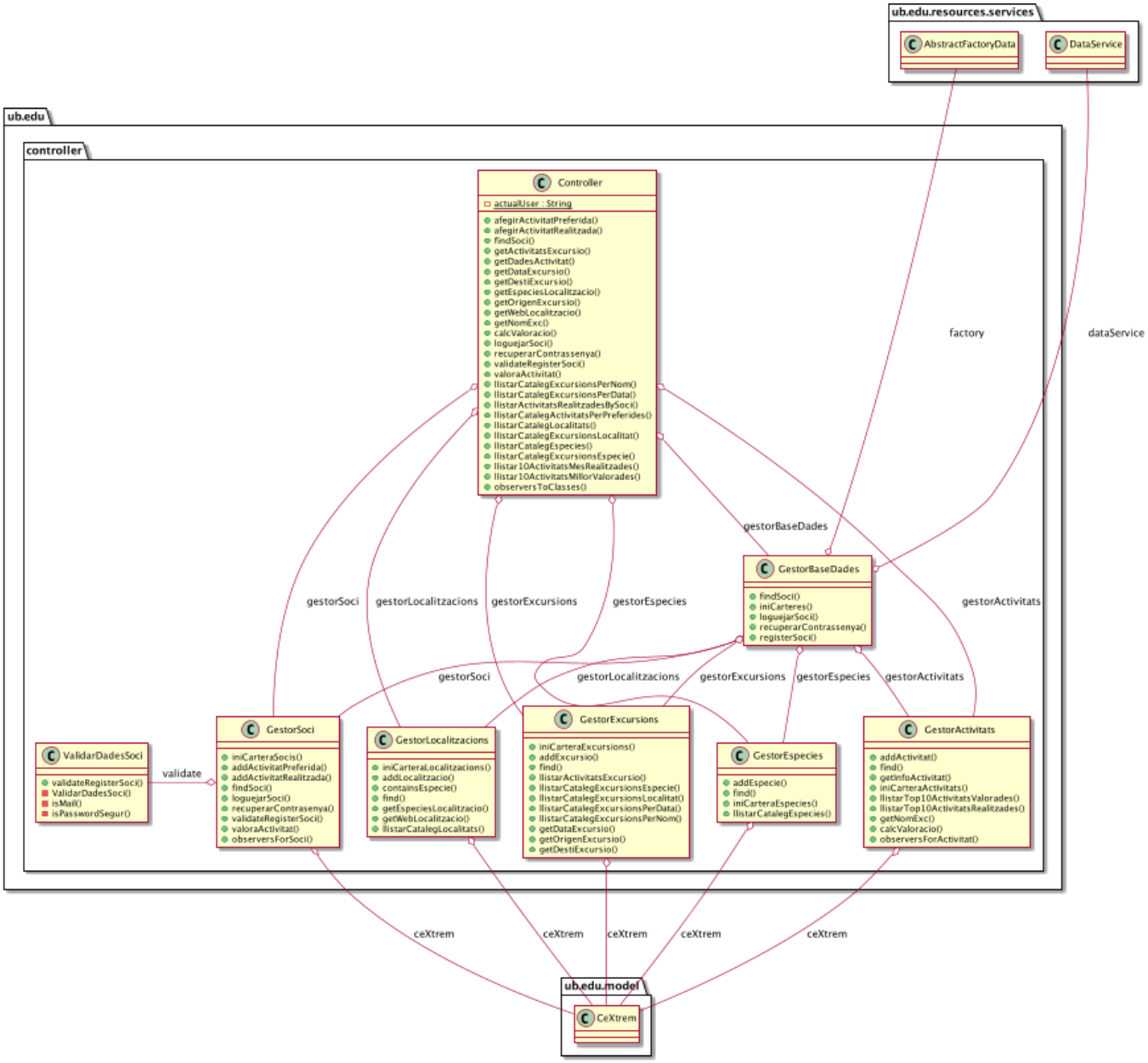

Figure 4.4: Controlador

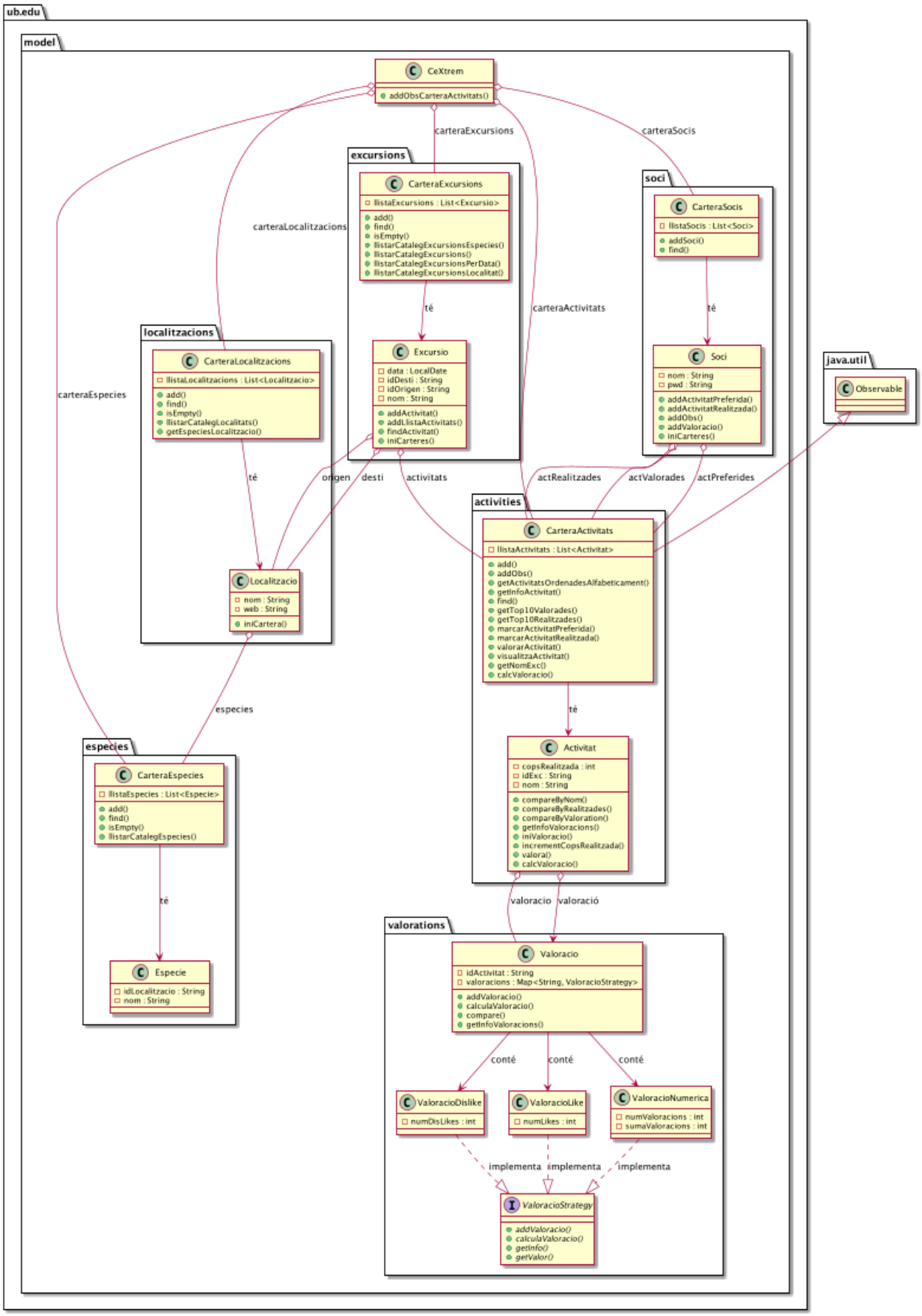

Figure 4.5: Model

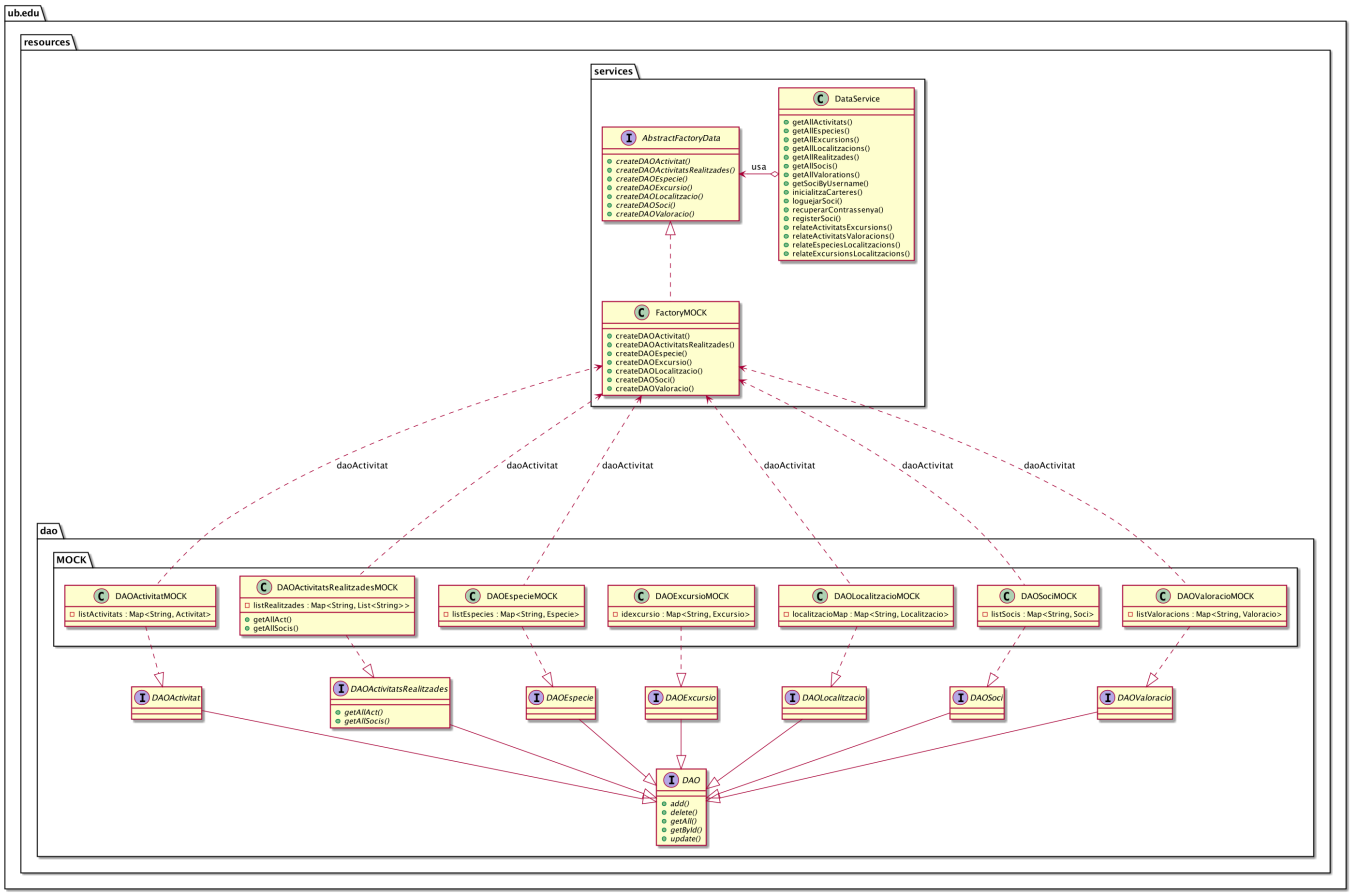

Figure 4.6: Capa Resources

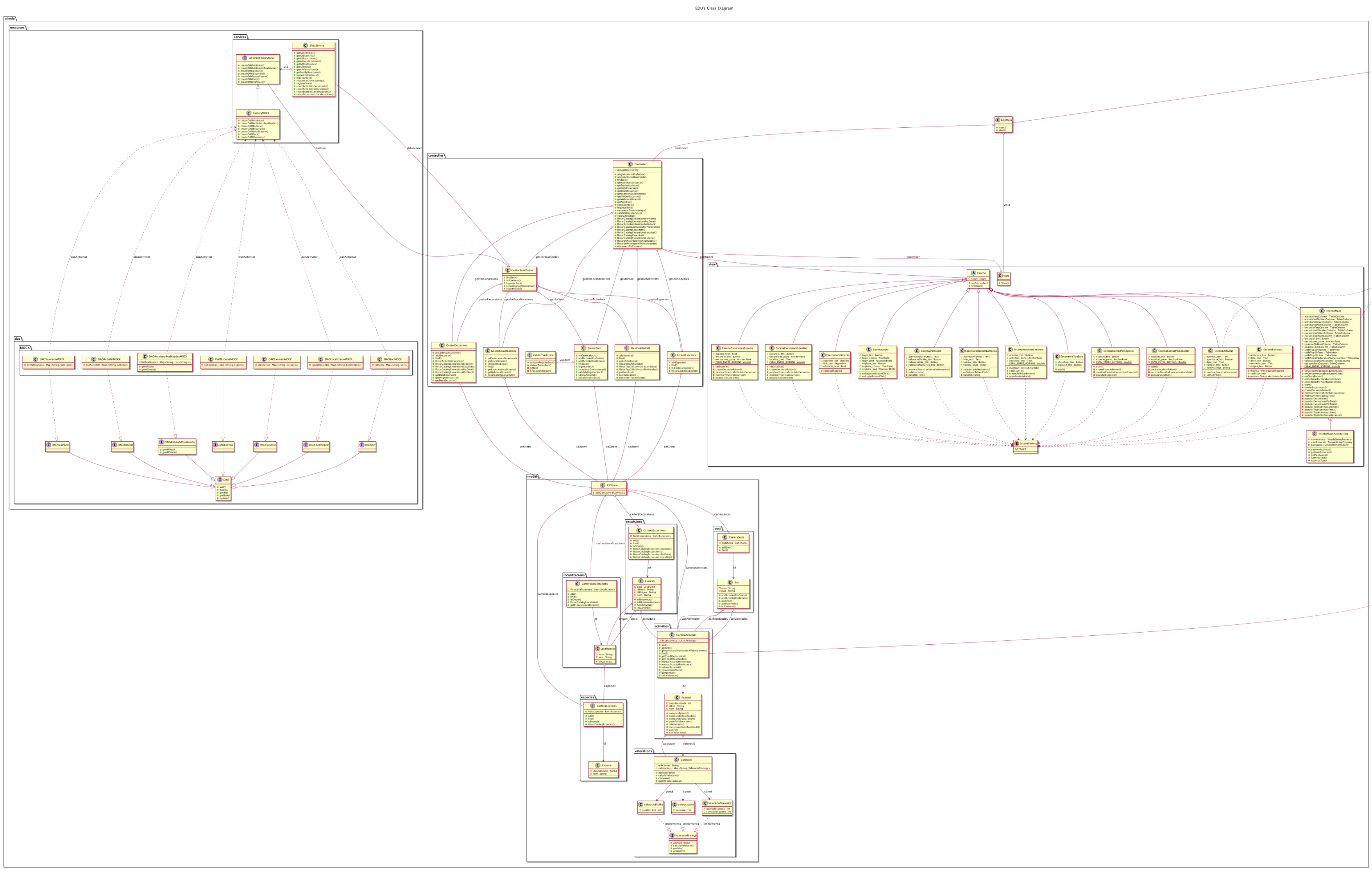

Figure 4.7: Diagrama de Classes General Pràctica 4

5 Diagrama de casos d'ús

Adjuntem imatge dels canvis realitzats al diagrama de casos d'ús de la nostra pràctica:

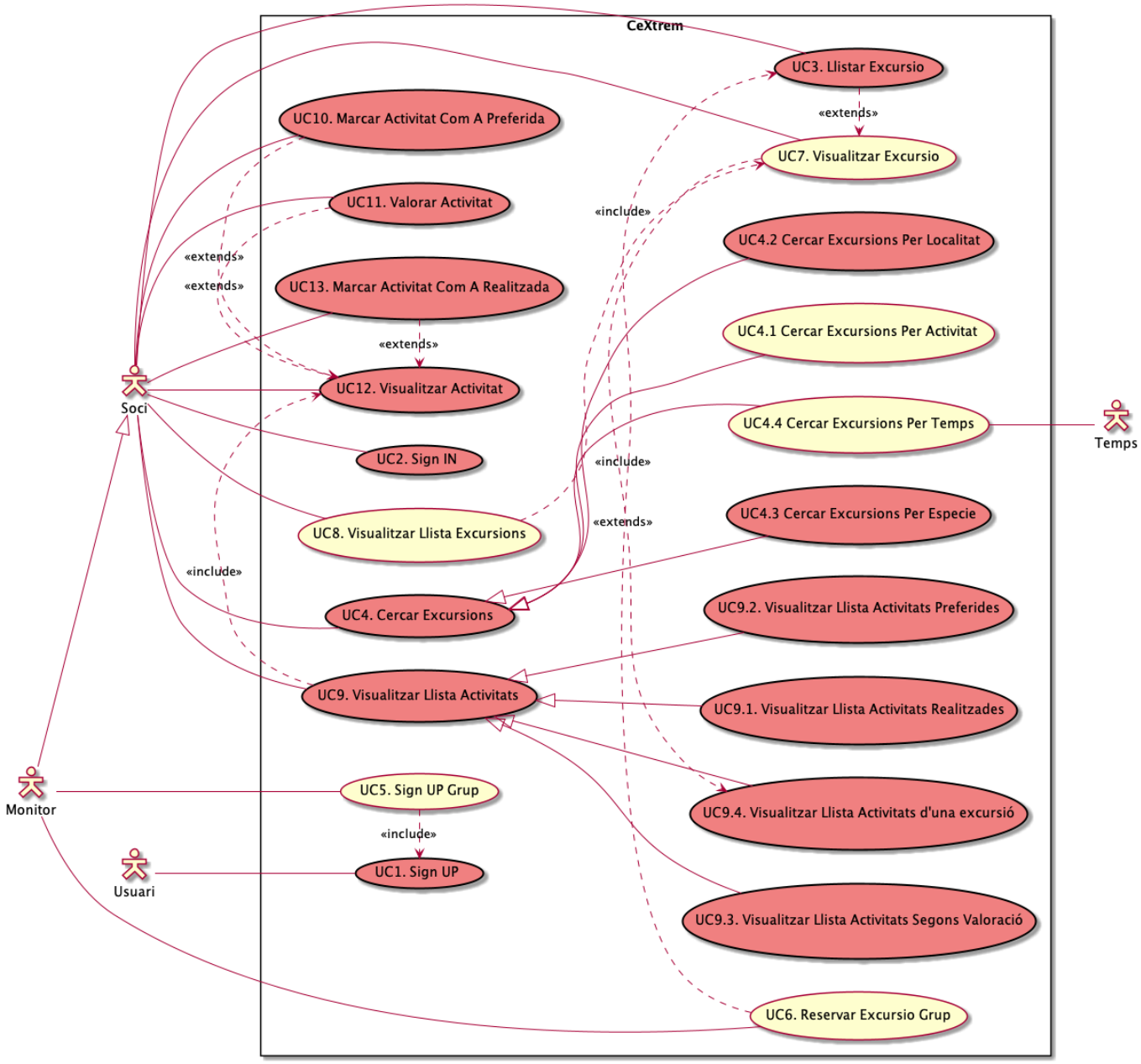

Figure 5.1: Diagrama de casos d'ús P4

6 CONCLUSIONS & OBSERVACIONS GENERALS

En aquesta última pràctica de l'assignatura hem aprofitat per acabar de perfeccionar les pràctiques anteriors per fer-les encara més estables i a més hem aconseguit implementar tant una interfície gràfica com el patró *Observer* explicat a les classes de teoría.

A més dels mínim productes viables necessàris per a poder treure la màxima nota, vam considerar adient realtizar els opcionals ja que volem optar a la màxima nota possible. Així doncs, hem implementat les dues opcions de poder cercar excursions tant per espècies com per llocs, la possibilitat de veure el Top10 activitats amb més likes i també la possibilitat de veure el Top10 d'activitats amb més dislikes.

També hem afegit algunes característiques adicionals que considerem milloren l'experiència d'usuari a l'utilitzar la nostra aplicació. Entre aquestes es troben que si fem clic al botó de tancar l'***EscenaMain*** se'ns tancaran la resta de finestres i ens apareixerà una finestra demanant-nos si volem sortir de l'aplicació o bé fer un *Sign out*, per tal d'iniciar sessió amb un altre usuari. Per tal de poder tancar totes les finestres al tancar la finestra principal, hem hagut d'ajudar-nos d'una caracerística de *Java* que ens permet indicar als diferents *Stages* qui és el seu propietari (*Owner*). A totes les finestres que es poden arribar a obrir després d'haver accedit a l'***EscenaMain*** l'hem assignat com a propietari aquesta escena, ja que és la principal de totes. Per tal de dur a terme aquesta funcionalitat, hem hagut d'afegir un altre constructor a ***EscenaFactory*** per tal d'assignar un propietari a un *Stage*.

Hem actualitzat bastants fitxers respecte a pràctiques anteriors, entre d'altres es troben el model de domini, el diagrama de casos d'ús i els diagrames de classes (tots ells es troben en aquest informe i a la carpeta *doc* de la pràctica. Però també hem aprofitat per actualitzar les històries d'usuari (*Excel*), i també l'hem guardat a la carpeta esmentada anteriorment.

Hem treballat tots dos de manera conjunta durant tot el transcurs de la pràctica, hem fet reunions presencialment i de manera online per tal de discutir els diferents aspectes de la pràctica. Creiem que hem treballat perfectament com a equip durant totes les pràctiques realitzades.

Volem acabar dient que aquestes pràctiques han estat molt profitoses per tal de veure en funcionament tot el que hem après a les classes de teoría, ja que tot queda molt més clar quan ho intentes per tu sol i has de ser capaç d'entendre i aplicar tot el que t'han ensenyat. Considerem que el format de fer 4 pràctiques amb una certa continuïtat és molt adient, i es fa molt més amè. Esperem poder fer servir tot el que hem après tant a l'assignatura com amb la realització d'aquestes pràctiques.

BONES FESTES!