Pràctica 2: Grups A i B

Objectius de la Pràctica 2:

L'objectiu principal d'aquesta pràctica és avaluar el codi desenvolupat a la pràctica 1 usant els principis disseny GRASP, SOLID i el patró arquitectònic per capes integrant l'arquitectura clàssica de Model-Vista-Controlador.

Com a objectiu secundari, s'aprendrà a utilitzar el patró DAO per a codificar la capar de recursos.

Així, aquesta pràctica consta de dues parts:

1. Analitzar i refactoritzar el codi sota la perspectiva de l'acoblament i la cohesió de les classes que has desenvolupat i implementat a la pràctica 1
2. Permetre la inicialització de les dades a partir de diferents recursos: d'unes classes que "simulen" una base de dades (DAO-MOCK)

Per a realitzar aquesta pràctica es seguirà el Model-Vista-Controlador explicat a classe de teoria, desenvolupant la part del Model i del Controlador. La part de la Vista es fa a partir dels tests de Concordion. Així, l'output d'aquesta pràctica no serà mitjançant la consola o una finestra gràfica, sinó que serà mitjançant els tests d'acceptació que vosaltres heu programat amb Concordion. Les parts de Controlador i Model s'han d'implementar en dues carpetes separades (controller i model, respectivament) dins de la carpeta src del projecte. La part del projecte que té relació amb la capa de recursos es guardarà en una carpeta interior a src, anomenada resources.

Punt de partida:

Seguint amb les funcionalitats relatives a monitoritzar un track quan es fa una ruta i poder fer un ranking del grup que ha fet el mateix track, es suposa que teniu la següent llista de Històries

d'Usuari implementades (potser no son exactament les històries que tens proposades però en tens de similars):

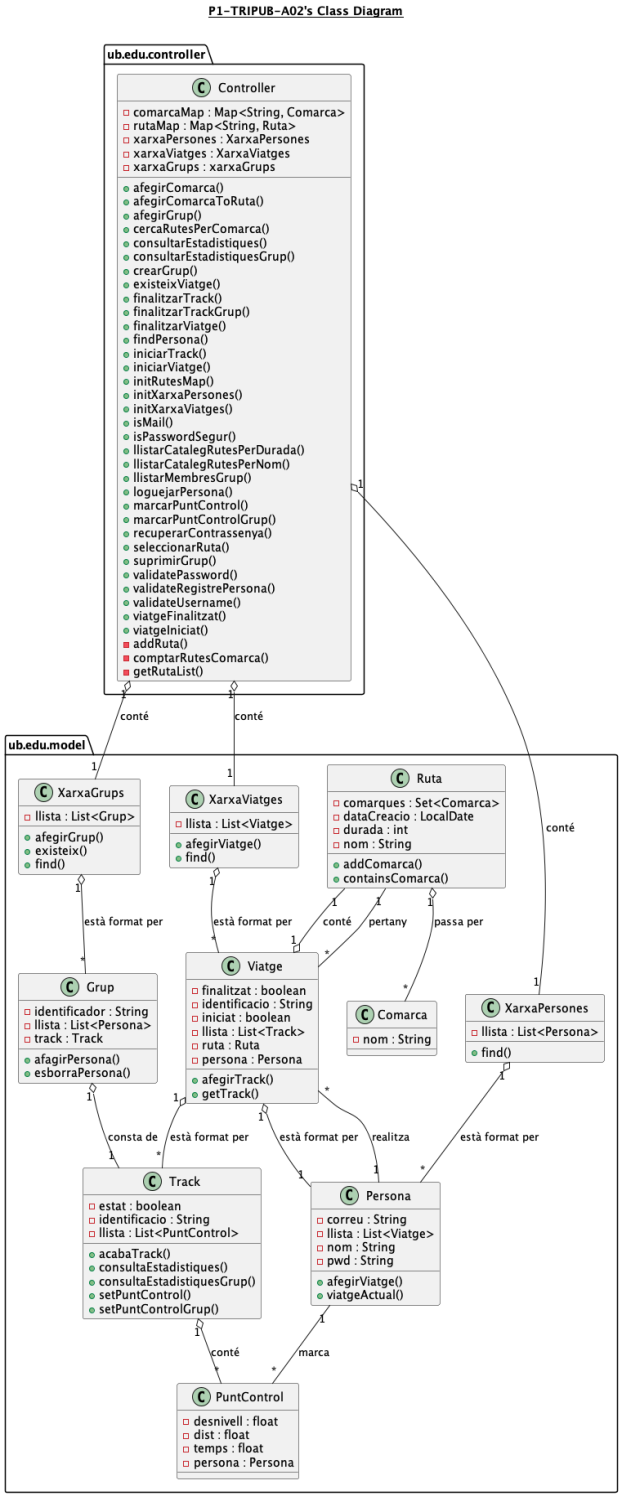
- Històries UC1. Enregistrar-se
- Històries UC2. Login
- Històries UC3. Llistar Rutes
- Històries UC4. Cercar Rutes per Comarques
- Històries UC5. Seleccionar Ruta
- Històries UC6. Començar Ruta
- Històries UC7. Finalitzar Ruta
- Històries UC8. Començar Track
- Històries UC9. Afegir Punt de Control a un Track
- Històries UC10. Finalitzar Track
- Històries UC11. Mostrar Estadístiques d'un Track
- Històries UC12. Crear Grup
- Històries UC13. Afegir-se al Grup
- Històries UC14. Marxar del Grup
- Històries UC15. Calcular Rankings de Grup d'un Track

PAS 1: Inspecció del codi – Usant dels criteris GRASP

Segueix els següent passos i contesta els següents punts:

1. Extreu el diagrama de classes del teu codi de la pràctica 1 utilitzant el plugin Sketch it! de IntelliJ (Menu Tools). Completa el fitxer plantUML per a detallar totes les dependències entre classes. Inserta aquí la imatge del diagrama de classes del teu projecte:

TripUB : App per mòbil per planificar fàcilment viatges i rutes



2. Detecta males olors del teu codi:

a. Identifica quines classes estan més acoblades amb d'altres. Llista-les a continuació:

L'acoblament és una mesura del grau de connexió, coneixement i dependència d'una classe respecte d'altres classes. Una classe està mal acoblada quan té un alt acoblament.

Mirant l'anterior diagrama de classes, es veu clarament que Viatge és una classe molt mal acoblada, ja que depèn de moltes classes, de Persona, de Ruta i de Track... A més és l'encarregada de crear Punts de Control entre d'altres, i no hauria de ser-ho

La relació entre Persona i Punt de Control es podria treure, redissenyant una part del codi, i faria baixar l'acoblament en aquestes dues classes, que no han d'estar relacionades.

El mateix passaria amb Track i Viatge, ja que Viatge no hauria de contenir Tracks en general. Gran part del problema d'aquesta part consisteix en no haver separat Track i Punts de control en si es tracten de manera individual o grupal.

Controller depèn també de moltes classes, que fa que tingui més acoblament del que hauria. A més, crea persones, viatges, rutes, grups... No hauria de ser la classe encarregada de fer-ho.

En resum, hi ha moltes classes que fan molta més feina de que haurien, ja que no haurien de tenir coneixement respecte algunes classes.

b. Identifica quines estan menys cohesionades. Llista-les a continuació:

La cohesió és una mesura del grau de relació i de concentració de les diverses responsabilitats d'una classe (atributs, associacions, mètodes, …). Una classe està mal cohesionada quan té una cohesió baixa.

La classe Controlador està clarament fatal cohesionada, té molts mètodes i fa moltes funcions diferents. Es podria, per començar, per exemple crear una classe que servís per inicialitzar el Controlador.

La classe Track té mètodes que no li correspondrien, per exemple, calcular les estadístiques d'un Grup. Com a més fa funcions com marcar punts de control, es pot dir que és una classe també molt mal cohesionada.

Persona té atributs que no li pertoquen, i el fet que Track tingui dues inicialitzacions diferents ja fa pensar que també és una classe mal cohesionada.

PAS 2: Refactorització del codi usant TDD – Aplicant dels criteris GRASP

Segueix els següent passos, tot contestant els punts que ho requereixin:

1. Clona el projecte base de la pràctica 2 que tens en el classroom. En el projecte base es proporcionen els mateixos tests de la pràctica 1 base.

   https://classroom.github.com/a/_oV_GMAk

2. Obre el teu projecte de la pràctica 1 i també el codi que acabes de clonar. La idea es que vagis passant el codi, test per test, que ja tens a la pràctica 1 al nou projecte que et donem i **refactoritzant** el que sigui necessari del teu projecte anterior, per deixar-lo en el projecte nou sense males olors. **Deixa per ara les inicialitzacions de les teves dades tal i com les feies a la pràctica 1, sense usar el patró DAO (excepte per les històries d'usuari que ja et venen implementades en el nou projecte).**
   a. Dedica't a UN test d'acceptació per començar a veure com es comporta el codi internament en el teu diagrama de classes anterior.
   b. Intenta seguir les recomanacions GRASP per poder millorar el codi d'aquest test. Recorda que els patrons GRASP els pots trobar a les transparències de classe de teoria (Tema 3.3): Baix Acoblament i Alta Cohesió, Expert en Informació, Creador, Controlador, Fabricació Pura, Indirecció, Polimorfisme i Variacions Protegides. Els tens explicats també en el vídeo 18 – Patrons GRASP. No vol dir que hagis d'usar tots els patrons a cada test, sinó potser et serveixen de guia per poder modificar el codi.
   c. Refactoritza i inclou el nou test en el nou projecte (el facilitat en el classroom de la pràctica 2) segons els criteris que pensis i prova el test via Concordion i tots els tests anteriors per a validar que segueixen funcionant. Potser que ara no vagi algun dels tests posteriors que estan en el mateix html que has inclòs en el nou projecte, però fixa't només en els que vagis refactoritzant. Omple la taula que tens a continuació per detallar per aquest test, els criteris GRASP que has usat.

d. Puja el test que ja funciona al github i posa en el commit el missatge que identifica el test correctament i el patró o patrons GRASP que has usat.
e. Es procedeix a refactoritzar el següent test d'acceptació (anar al pas a.).

Així, per cadascuna de les funcionalitats demanades, haureu de fer un add/commit/push a github per cada test d'acceptació implementat amb el comentari adient del test i quin patró GRASP heu usat. Això vol dir, que en finalitzar l'entrega haureu de tenir, com a mínim tants commits/push a github com tests d'acceptació. Normalment, es procedeix a fer un commit/push a cada test d'acceptació per a no acumular molts canvis del projecte local en relació al remot. Tal com s'esmentava en el pas c., omple la taula que tens a continuació indicant per a cada test d'acceptació, el criteri GRASP que has fet servir i explica breument què has canviat en el teu codi inicial. Afegeix files a la taula si així ho necessites:

Test d'acceptació	Criteris GRASP	Breu explicació de les classes canviades
UC5 – Seleccionar Ruta (tots els seus tests)	- Expert en informació - Creador - Acoblament	A part de canviar els tests per no fer-los tots simples, he fet que el mètode de la classe Controlador encarregat de seleccionar el viatge delegués part de la informació, com crear el Viatge, a la classe XarxaViatges, ja que hauria de ser ella l'encarregada.
UC6 – Iniciar Ruta (tots els seus tests)	- Expert en informació - Controlador - Cohesió	La classe Controlador interectuava amb el viatge directament iniciant-lo, i no és responsabilitat seva,

		aleshores, aquest mètode s'ha passat a fer a XarxaViatges.
UC7 – Finalitzar Ruta (tots els seus tests)	- Expert en informació - Controlador - Cohesió	Anàlog al UC6, delegant responsabilitats també.
UC8 – Iniciar Track (tots els seus tests)	- Expert en informació - Controlador - Cohesió	Tal i com en els dos anteriors UC, el Controlador no ha de ser la classe encarregada d'activar un Track, ho és la classe Viatge, així que modifiquem el codi perquè sigui ella la classe responsable. Haurem de passar però per les classes Controlador i XarxaViatges
UC9 – Marcar Punt de Control (tots els seus tests)	- Expert en informació - Acoblament	Deleguem la responsabilitat de marcar i crear un punt de control a la classe Track, i no al Controlador com ho teníem implementat inicialment.
UC10 – Finalitzar Track (tots els seus tests)	- Expert en informació - Controlador - Cohesió	Anàlog al UC8, delegant responsabilitats també.
UC11 – Mostrar Estadístiques de Track (tots els seus tests)	- Expert en informació - Controlador - Cohesió	UC11 i UC15 ho teníem com un sol cas d'ús en la pràctica anterior. Així, inicialment hem separat els tests en dos diferents.

		El càlcul de les estadístiques del Track es feien una part a Track, una part a Viatge i una part a Controlador. Hem modificat el codi perquè es facin integrament a Track, ja que és ell la classe experta.
UC12 – Formar Grup (tots els seus tests)	- Expert en informació - Polimorfisme - Creador - Acoblament - Fabricació Pura - Indirecció	UC12, UC13 i UC14 ho teniem com un sol cas d'ús en la pràctica anterior, així que hem separat els tests, i els hem millorat fent-los en forma de taules. Per no haver de fer que Persona tingués una llista de Tracks actius, hem fet una herència de Track amb TrackIndividual i TrackGrupal i de PuntControl amb PuntControlIndividual i PuntControlGrupal, assignant a cada Track una llista dels Punts de control corresponents. Així s'ha baixat molt l'acoblament, ja que s'han pogut treure relacions entre Track, PuntDeControl, Grup i Persona. També, s'ha millorat molt el fet que un Track o un

		Punt de Control actués de manera diferent depenent de si estàvem a un grup o no. Ens hem donat compte més endavant que es podia eliminar la classe PuntControlIndividual, ja que no afegia ni atributs ni mètodes.
UC13 – Afegir-se a Grup (tots els seus tests)	- Expert en informació - Polimorfisme - Creador - Acoblament	Anàlog al UC12, millorant els tests per fer-los en forma de taules i fent una delegació de responsabilitats.
UC14 – Marxar de Grup (tots els seus tests)	- Expert en informació - Polimorfisme - Creador - Acoblament	Anàlog al UC12, millorant els tests per fer-los en forma de taules i fent una delegació de responsabilitats.
UC15 – Calcular Rànquing (tots els seus tests)	- Expert en informació - Controlador - Cohesió	Ha sortit bastant directe de manera correcte, ja que els canvis que hem fet en el UC11 (encarregar a TrackIndividual calcular les estadístiques recorrent els punts de control) són anàlogs en aquest cas d'ús, demanant-ho a la classe

		TrackGrupal recorrent els seus punts de control grupals. Hi ha alguna altre petita modificació que s'ha hagut de fer, per quadrar bé la herència.

3. Detalla el **diagrama final de classes** que has obtingut al final de la refactorització de tots els tests d'acceptació. Si detectes encara alguna "mala olor" en el codi", explica-la aquí breument:

Introdueixo només el diagrama de classes de la part de controlador i model, ja que la part de Service/MOCK/DAO no les hem tocat (de moment) i no ens aportaràn informació útil per veure l'eficéncia dels canvis que hem aplicat.

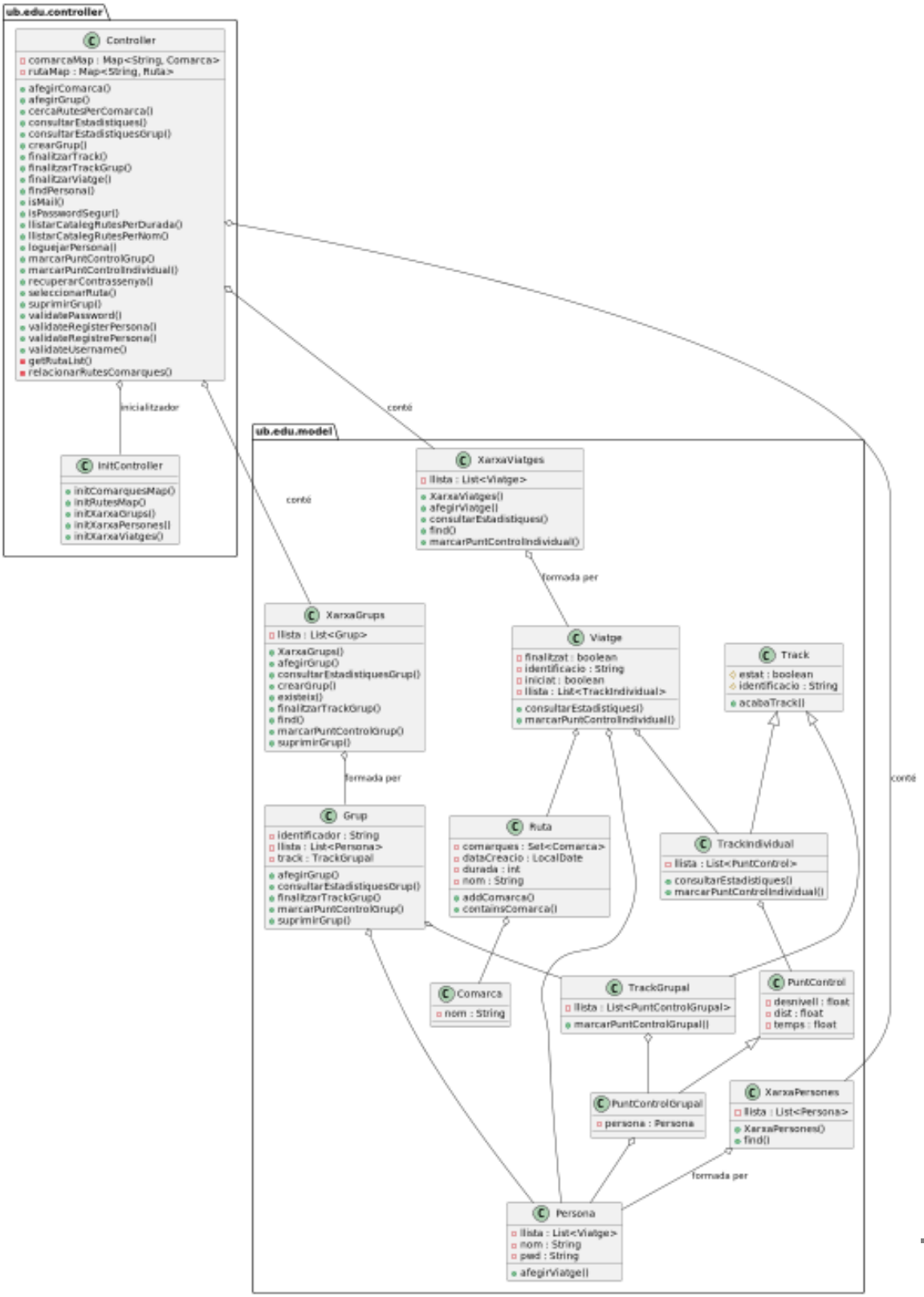
Veiem que s'ha baixat en general l'acoblament, baixant la dependència entre classes, i s'ha augmentat la cohesió, reduïnt el nombre de mètodes i responsabilitats de cada classe, reassignant i delegant les resposabilitats.

La classe InitController ens ajuda a treure-li responsabilitats d'inicialitzar les dades al Controller, el qual però segueix tenint moltíssims mètodes.

Les classes Controlador i Viatge segueixen tenint un alt acoblament, amb l'ajuda d'alguns patrons com el patró façana es podria reduir l'acoblament.

Que una classe tingui un mètode no vol dir que sigui l'encarregada de resoldre'l. Per exemple, marcarGrupDeControlGrupal() es resol a TrackGrupal, però el mètode ha de navegar per Controlller, XarxaGrups i Grup fins a arribar a TrackGrupal, això pot semblar quan es veu al diagrama que proporciona una cohesió baixa a les classes intermitges.

TripUB : App per mòbil per planificar fàcilment viatges i rutes



PAS 3: Utilitzant la capa de recursos: Patró DAO

En aquesta part, aprendrem a utilitzar i codificar el patró DAO per inicialitzar les dades de l'aplicació usant de forma flexible recursos externs com poden ser fitxers o bases de dades. Per això, segueix els següent passos i contesta els punts que ho requereixin:

1. Explora com el projecte s'estructura en una arquitectura per capes:

    - El software es basa en una arquitectura en tres capes: (1) la capa de vista que és la que correspon als tests, (2) la capa de lògica de negoci (on està el model i el controlador) i (3) la capa de recursos o persistència, que és l'encarregada de guardar les dades. La capa de recursos o persistència es podrà substituir per l'accés a una bases de dades compartides amb tota la classe amb dades sobre les excursions, activitats, socis, etc.. Quan s'iniciï l'aplicació, s'inicialitzaran totes les dades de la capa de lògica de negoci necessàries per a l'aplicació.
    - En el repositori de github es proporciona un codi de la part de persistència (o recursos) que cal utilitzar i ampliar. En aquesta part s'utilitzen els patrons de disseny d'AbstractFactory i DAO per a poder canviar fàcilment la capa de persistència.
    - En aquest projecte base s'inicialitzen les dades des d'un MOCK de la Base de Dades (un *mock* és un objecte que simula un altre objecte a efectes de test). En el projecte base que us proporcionem hi han quatre Mocks, un per inicialitzar Persones, un altre per inicialitzar Rutes, un altre per inicialitzar Comarques i finalment un quart per inicialitzar les relacions entre Rutes i Comarques.
    - En el codi que s'us facilita es considera (o pretén simular) que l'aplicació podrà estar offline (sense connexió externa) i per tant, totes les dades hauran d'inicialitzar-se a memòria i no des d'una base de dades, per ara. Per donar aquesta utilitat de poder "simular" la base de dades, es fan servir classes MOCK.
    - Trobareu una descripció més detallada en el Manual del Campus Virtual.

2. En relació al codi de la capa de persistència (carpeta recursos i data service), quins patrons GRASP s'estan utilitzant?. Llista'ls a la taula que tens a continuació, justificant breument la teva resposta (afegeix files a la taula si és necessari):

Patró GRASP usat a la capa de persistència	Breu Justificació
Cohesió alta a les classes DAO-MOCK	Les classes DAO-MOCK internament només tenen la responsabilitat de gestionar les dades com si fossin una base de dades i cap altra
Acoblament baix	La capa de persistència desacobla la capa de negoci amb la capa de dades.
Creador	El creador de les instàncies és la classe que té coneixement d'elles, és a dir la «base de dades», la classe FactoryMock
Indirecció	Per fer de pont entre tots els DAO-Mock i la «base de dades» utilitzem la classe FactoryMock.
Controlador	Tenim com a controlador d'aquesta part la classe DataService, que es comunica amb el Controller del model i és la responsable de fer les crides.

3. Modifica aquest projecte per inicialitzar localitzacions en el seu DAO corresponent per poder fer una Cerca de Rutes per localitzacions (en lloc de comarques). Quines classes caldria que actualitzessis? Per això intenta resseguir el codi des del Controlador per veure com es fa la inicialització de les comarques i el seu lligam amb rutes. Intenta modificar el codi per donar d'alta les Localitats. Explica aquí breument la teva solució.

Hauríem de crear les classes Localització i DAOLocalitzacióMOCK (emulant DAOComarquesMOCK) i DAORelacioRutaLocalitat (emulant DAORelacioRutesComarques).

També, crear les interfícies DAOLocalitzacio i DAORelacioRutaLocalitat.

Haurem de modificar factoryMOCK, dataservice i AbstractFactoryData per tal d'incloure els mètodes que porten l'informació de la DAO fins al controlador. Dins el controlador haurem de crear un mètode per inicialitzar les dades de les Localitzacions i relacionar-les amb les rutes, així com incloure un mètode per afegir localitzacions i per extreure les rutes relacionades prèviament amb aquestes localitzacions.

4. [OPT] Integra el codi que gestiona la capa de persistència a la teva pràctica. Afegeix els DAOs corresponents a noves dades que necessites. Com gestionaràs la càrrega de Grups de Persones? Com gestionaràs el progrés d'un track d'una Persona? Raona la teva solució a continuació, justificant-la segons els patrons GRASP. Vigila d'integrar poc a poc, test a test, els mètodes del controlador per a tenir la capa de persistència ben lligada.

Hauríem de crear un DAOGrupMOCK i un DAOGrup (interfície) que rebi una llista de persones (potser desde DAOPersonaMOCK) i després passés aquestes dades (mitjançant el DataService) al controlador per tenir un registre (com una mena de base de dades) de tots els grups que existien abans que l'aplicació quedés "offline". Pel track, potser podríem incloure un DAOTracksMOCK i un DAOTracks (interfície) que es relacionessin amb cada persona que ha fet aquest Track a través de un DAORelacioPersonaTrackMOCK i DAORelacioPersonaTrack de la mateixa forma que està implementat amb Ruta i Comarca, que es passés al controlador i aquest s'encarregués, com amb Ruta i Comarca, de relacionar-les i així poder guardar registre dels Tracks que ha fet cada persona.

PAS 4: Repartició de feina i comentaris finals

Finalment inclou aquí el repartiment de la feina que heu decidit per realitzar aquesta pràctica:

Víctor i Guillem:

- Inicialment, els dos hem fet conjuntament el diagrama de classes de la pràctica 1, i hem analitzat el codi de la pràctica, detectant les males olors.

- A més, sempre hem comentat el que ha fet l'altre persona, ho hem revisat i ens hem ajudat en qualsevol tipus de dubte

Víctor Sort:

- Refactorització UC5, UC6, UC7, UC8, UC10, UC11

- Redacció d'aquest document i resposta a les seves preguntes

Guillem Navarra:

- Refactorització UC9, UC12, UC13, UC14, UC15

- Implementació exercici 3.3 (DAO Localitat)

Instruccions per al lliurament

Tots els lliuraments es presenten junt amb una còpia d'aquest document amb les respostes incloses a cada pregunta.

El dia del lliurament has de pujar al campus virtual **dos fitxers**: (Només pugeu un lliurement per equip)

- Aquest fitxer exportat a format pdf
- Un fitxer zip amb el projecte solució que correspon al projecte que tens en el teu repositori de github. Marca el darrer commit del github com VERSIO FINAL PRACTICA 2