L6 -JPA Partea a 3-a: EntAppClient

Mai multe detalii pentru laboratorul trecut gasiti in cele 2 pdf-uri:

1. Detalii JPA.pdf
2. Essential object-relational mapping in JPA.pdf

EntAppClient

Acum sa vedem cum putem sa accesam in programe Java datele din baza de date.

Pentru intelege mai bine ce vrem, sugerez sa aruncati o privire rapida pe documentatia unei aplicatii "roster": http://iscoreleagues.com/manual/. Este o aplicatie pentru Androi/iOS, si ne imaginam ca vrem sa o implementam ca o aplicatie JavaEE.

Foarte important de inteles ca e doar un exercitiu: in realitate cerintele aplicatiei nu le am date ci se obtin treptat, iterativ, de la stakeholderi ("A party that has an interest in an enterprise or project").

Oricum din documentatie rezulta evident cateva concepte specifice domeniului aplicatiei (business objects)

"*Well, a Business Object is generally considered to be a class that represents an Entity, e.g. a Book or a Store. Such a class has certain properties like price, colour, width, isbn number etc. In Java or .NET, it consists of so-called setters and getters, i.e. methods that set or get those properies.*

The Business Logic on the other hand is that part or a programm (that layer) that works with those properties, i.e. how is this book sold. The business logic layer uses the business objects in order to access the database."

Cateva dintre concepe: Player, Team, League corespund cu cele din aplicatia "roster" pe care o studiem noi. De asemenea rezulta imediat cateva responsabilitati ale aplicatiei: adaugare/stergere jucatori, echipe ligi. Undeva in aplicatie probalil am nevoie de motode de genul: getAllPlayers, grtAllTeams, addPlayer, removeTeam, etc. Si aceste responsabilitati se deduc, in general, iterativ pe masura ce studiem un *Use Case*, eventual cu ajutorul ***SSD*** (System Sequence Diagram discutata la curs). Problema alegerii clasei care ofera o anumita resposabilitate (de exemplu in ce clasa se afla metoda getAllTeams() ?), este mai complicata. Acum, simplificam: presupunem ca vrem sa implementam anumite metode si sa testam daca fac ce trebuie (business logic). In aplicatia pentru Android referita mai devreme, probabil ca getAllTeams() este aplelata pentru afisarea tuturor echipelor intr-o componenta Android. In aplicatia pe

care (ne imaginam ca o) proiectam noi, putem sa folosim acelasi stil de afisare sau nu. De fapt nici nu am stabilit ce tehnologie folsim pentru GUI: JSP, SWING, JSF,…

Recomandare de la Oracle:

'' Avoid building all your model's descriptors in a single iteration. Start with a small subset of your classes. Build and test their descriptors, then gradually add new descriptors and relationships. This lets you catch common problems before they proliferate through your entire design.

Write Java code to use database sessions. Sessions are used to query for database objects and write objects to the database"

Practic trebuie sa:

- Cream o clasa de tip EJB(Enterprise Java Bean). In czul nostru ea este deja scrisa: "RequestBean" din proiectul roster-EJB
- In aceasta clasa scriu toate metodele necesare si pe care vreau sa le testez. Acceptam metodele deja de catre autorii aplicatiei roster: addPlayer(), createLeague(), etc
- Cream o clasa: la noi RosterClient din proiectul roster-appClient. In metoda **main()** din aceasta clasa avem secventa care testeaza metodele business (din clasa RequestBean)

Este destul de greu de inteles in acest moment cum se realizeaza comunicarea intre cele doua componente: RosterClient (ruleaza la client) si RequestBean(ruleaza pe serverul Java EE GlassFish). Cheia o constitue faptul fiecare dintre ele ruleaza sub controlul unui container specific iar cele doua containere asigura, transparent pentru programator, aceasta comunicare:

- RosterClient intr-un container client
- RequestBean ruleaza in containerul EJB

Admitem pentru moment, fara prea multe explicatii, cateva reguli:

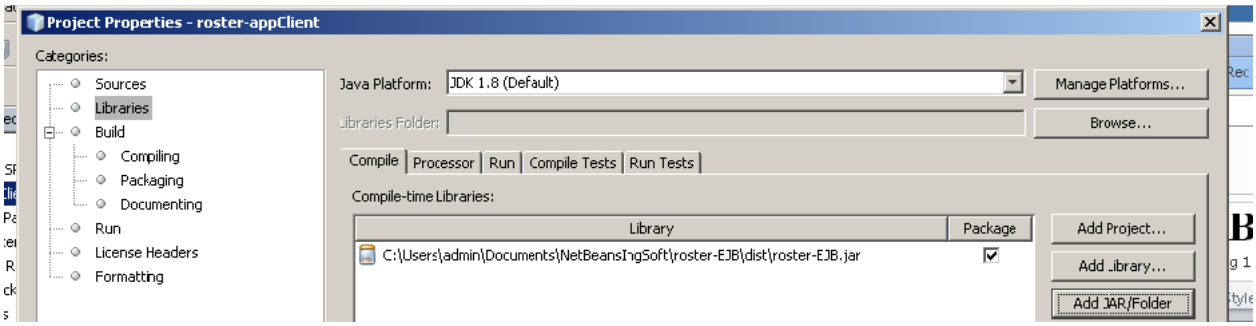
- In aplicatiile care fac acces la baze de date, obiectele "entity" se manipuleaza doar prin intermediul interfetei **EntityManager**. Cum se utilizeaza principalele metode ale obiectului EnityManager se poate observa in codul din aplicatia studiata (in clasa RequestBean)
- Referirea la EntityManager se face dintr-o clasa **EJB**(pentru moment, numim EJB o clasa normala java(POJO i se mai zice), atat ca este adnotata cu **@Stateful** sau **@Stateless** (in general nu-I acelasi lucru dar in exemplu nostru se poate folosi oricare dintre ele). De remarcat ca obiectul "em" care joaca rolul de EntityManager este doar declarat(EntityManager em;), apoi folosit, dar NU este instantiat nicaieri in codul nostru. Magia o constitue adnotarea ***@PersistenceContext*** : atunci cand clasa RequestBean este instantiata, se injecteaza automat, in variabila **em,** o referinta spre un obiect EntityManager(acesta este configurat cu "persistence.xml"). Mai mult, nici RequestBean nu este instantiata de catre noi. O instanta a acestei clase este oferita de containerul EJB atunci cand, in aplicatia client, RosterClient, se cere " injectarea" unui astfel de obiect prin adnotarea **@EJB** din clasa RosterClient

- Aplicatia client ruleaza pe alt calculator (alt JVM). Pentru ca acest client sa poata comunica cu ejb-ul, clasa EJB mai trebuie sa implementeze o interfata java care la randul ei are o adnotare: **@Remote**. In rest, ca orice interfata Java, contine metodele care trebuie scrise in orice clasa care o implementeaza. In exemplul nostru, ejb-ul este RequestBean iar interfata este Request.

O posibilitate simpla de testare a metodelor din clasa RequestBean o ofera aplicatiile de tip *Client Java EE*. Pasii de urmat pentru crearea acestei aplicatii sun descrisi pentru Netbeans. (Aviz pentru Eclipsari: Am incercat in urma cu un an sa creez in Eclipse un astfel de client dar nu a functionat. Recunosc ca nu am insistat prea mult deoarece in Netbeans a mers din prima)

1. In NetBeans File->NewProject..->Java EE -> **Enterprise Application Client**
2. Next si introduc nume proiect: roster-appClient
3. Next si introduc pentru Main Class: roster.client.RosterClient. Finish
4. Copiez din cel de-al doilea proiect “roster-app-client”, codul din RosterClient.java in clasa cu acelasi nume care s-a creat in proiectul nostru
5. Compilam si ne uitam la erori

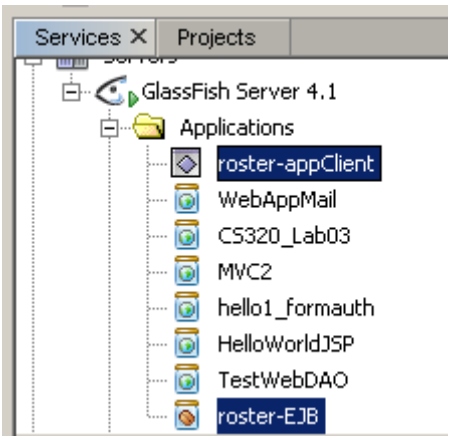
Probabil apar erori datorita referirilor la clasele din package-ul “roster.util” precum si la interfata Request. Acestea se gaseste in alt proiect (cel de data trecuta: roster-ejb). Pentru a avea acces la aceste clase in proiectul curent, in NetBeans tab-ul Projects, selectez proiectul client(roster-appClient) , cick-dreapta -> Properties. Apoi click pe Libraries. Click pe butonul “Add JAR/Folder”, in folderul unde creeaza Netbeans proiectele caut proiectul roster-EJB si aci, in directorul**dist** selectez jar-ul creat la compilare: “roster-EJB.jar”



Ok si “Clean and Buid” apoi “Deploy”.

ATENTIE: Serverul GlassFish trebuie sa fie pornit cand faceti Deploy

Dupa ce ati instalat (Deploy) cele doua proiecte, trebuie sa le regasesc in lista aplicatiilor instalate pe GlassFish. In Netbeans putem sa verificam:



Acum putem rulam aplicatia client: click-dreapta pe roster-appClient -> Run

Ca exercitiu util, implementati o functionalitate simpla din aplicatia Android/iOS aminitita la inceput. Si anume Merge Team: daca accidental am creat de 2 ori o echipa (exista 2 team-uri cu id-uri diferite dar au acelasi nume) doresc sa pastrez pe prima, sa adaug la prima ce contine in plus a doua echipa, si pe cea de a doua sa o sterg. Pentru testare creati o noua aplicatie client.

Un UseCase simplu in care nu se face nicio referire la GUI:

1. Clientul cere sistemului lista tuturor echipelor
2. Systemul raspunde cu aceasta lista
3. Clientul selecteaza cele doua echipe care au acelasi nume
4. Clientul crere systemului sa combine echipele selectate, cu id-urilr id1 si id2
5. Systemul returneaza noua lista

Exceptii:

4a. daca cele 2 echipe nu au nume identice se afiseaza eroare