4.1: درک entitie ها و value tpe ها:

زمانی که به مدل خود نگاه میکنید، به یک سری تفاوت میان کلاس ها دقت داشته باشید:

برخی از value type ها به نظر مهم می آیند؛ مثلا در مثال های قبل item,Category,User آبجکت هایی هستند که اهمیت بالایی دارند و معرفی کننده ی مقادیر متناظر مدل شما در دنیای واقعی هستند،

Typeهای دیگر در مدل ما مثل Address,String,Integer مقادیر کم اهمیت تر در مدل ما هستند؛

در این بخش، به بررسی تفاوت میان نوع موجودیت ها میپردازیم.

4.1.1: پیاده سازی مدل به صورت جزبه جز:

هدف اصلی هایبرنیت از پشتیبانی پیاده سازی جز به جز مدل رسیدن به یک مدل در نهایت قوی است. به همین منظور است که ما از pojo در کلاس مدل هایمان استفاده میکنیم.

در ساختار crude پیاده سازی ریز به ریز به معنای کلاس های بیشتر نسبت به جداول دیتابیس است.

برای مثال یک user ، ممکن است در مدل پیاده سازی شده مان دارای یک Address باشد؛ در عوض در دیتابیس ما ممکن است یک جدول برای USERS تعبیه کرده باشیم که دارای ستون های HOME\_STREET,HOME\_CITY,HOME\_ZIPCODE باشد.

ما در مدل ساخته شده مان ، ممکن است از یک رویکرد مشابه حتی استفاده کنیم ، یعنی مثل بالا که در دیتابیس آمدیم یک تیبل جدا برای USERS تعبیه کردیم با ستون های ذکر شده، بیاییم 3 تا property در کلاس user پیاده سازی کنیم؛ اما بهترین حالت همانطور که در بخش 1.2.1 قبلا هم مشکل را بررسی کردیم این است که در پیاده سازی مدل خود یک کلاس Address ایجاد کنیم و زمانی که هر USERای دارای آدرس خانه بود برایش جدول Address پر شود این نحوه معماری دیتابیس و مدل باعث reUseAbillity و سهولت در فهم جداول میشود و نحوه ی نوشتن query برای واکشی اطلاعات نیز بهتر میشود.

JPA همواره به استفاده ی کامل از پیاده سازی جز به جز برای مدل ها میکند.

مثال دیگر برای درک این تاکید این است که، اغلب people دارای یک email با مقداری String و در property ، USERS دارند.

یه رویکرد پیچیده این است که یک کلاس EmailAddress بسازیم و این کلاس دارای method و ساختار مجزا باشد مشکلی که در این حالت پیش می آید اسن است که در ساختار ORM که نحوه lifecycle و persist شدن شی به هم میریزد چرا که وابستگی ایجاد شده است.

بیایید با یک مثال به بررسی این موضوع بپردازیم ؛

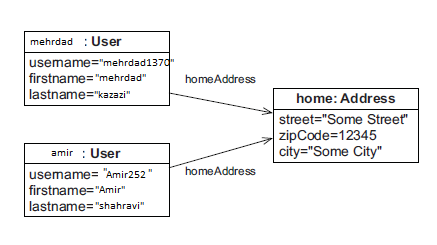
4.1.2: تعریف مفاهیم اپلیکیشن مورد نظرمان در واقعیت :

فرض کنید دو تا user داریم با یک آدرس مشترک که هر دو در این اپلیکیشن ما ثبت نام کرده اند،

به نام های امیر و مهرداد.

هر آبجگتی از user بیان کننده ی یک اکانت است. چون میخواهیم برای عمل ثبت نام save یا delete یا update انجام دهیم، پس شی user را به عنوان یک کلاس در نظرش میگیریم که در این کلاس، user ما دارای یک property به نام homeAddress است که جنسی از کلاس Address مان دارد.

در این حالت برای امیر و مهردادیک آبجکت Address داریم که به یک شی مشترک در کلاس Address ، references دارند.



در مدل بالا میبینید که دو تا user دارای آدرس های اشتراکی هستند، در این مدل اتفاقی که می افتد این است که شما نمیتوانید با پاک کردن user ، امیر آدرس آنرا نیز پاک کنید، چرا که lifecycle آدرس به User مهرداد هم وابسته است.

اکنون بیاید مدلمان را به این گونه پیاده سازی کنیم که هر user دارای یک Address مستقل باشد.

در این مدل دیگر User ها دارای آبجکت های مستقل هستند و در این صورت به راحتی میتوانیم هر user را به صورت مجزا پاک کنیم.

از این رو باعث میشود که هر آبجکت یه یک شی references داشته باشد مثلا با یک foreignKey به آبجکت یک کلاس دیگر وصل باشد و هر entity دارای lifecycle مستقل باشد.

4.1.3: تمایز قائل شدن بین entitieها و valueType ها :