Devoir Maison - Semaine 7

Domain-Driven Design – Partie 2

Vos réponses s'appuieront sur *Domain-Driven Design*, Eric Evans : chapitre 5 (sauf Modules et Modeling paradigms qui sont facultatifs), chapitre 6 et chapitre 7 (facultatif).

Pour la plupart des groupes, l'implémentation de l'application n'est pas encore très avancée. Dans les questions qui suivent, vous pouvez néanmoins répondre en vous appuyant sur un modèle et un design plus complets que ceux effectivement implémentés à ce jour. Il est préférable que ce modèle et ce design « cibles » soient élaborés en groupe, auquel cas vous pouvez aussi répondre aux questions en groupe (dans ce cas, merci de nous indiquer le nom du groupe).

1. Illustrer le principe « For every traversable association in the model, there is a mechanism in the software with the same properties. » en listant, pour chaque association dans votre modèle, sa contrepartie dans l'implémentation.

   Exemple :

   « *Plusieurs contrats d'assurance peuvent être souscrits par un client. On peut retrouver les contrats d'un client et on peut retrouver le client d'un contrat.* » (modèle)
   → « *Une instance de la classe* Policyholder *possède un membre* policies *de type* Policy[] *qui contient les contrats de ce client. Ce membre* policies *est initialisé avec le résultat d'une requête en BDD lors de l'instanciation de l'objet. Une instance de la classe* Policy *contient une référence vers une instance* Policyholder. » (design)

   ou bien (formulation équivalente pour le modèle ; solution technique différente pour le design) :

   « *Un client a une association one-to-many bidirectionnelle avec ses contrats.* » (modèle)
   → « *Une instance de la classe* Policyholder *possède une méthode* getPolicies *qui fait une requête en BDD et retourne un tableau de contrats de type* Policy[]. *Une instance de la classe* Policy *contient une référence vers une instance* Policyholder. » (design)
2. Prenons pour hypothèse que votre modèle contienne une liste de contrats vendus, et une liste des garanties possibles. Tous les contrats n'ont pas la même liste de garanties. Il y a une relation bidirectionnelle many-to-many entre les 2 : une garantie peut porter sur plusieurs contrats; un contrat peut comporter plusieurs garanties. La relation a-t-elle besoin d'être traversée dans les 2 sens ? Quel design choisissez-vous ?
3. Quels sont les entités, les « value objects » et les services de votre modèle ?
4. Pouvez-vous donner un exemple d'agrégat dans votre modèle ?
5. Donner un exemple d'invariant satisfait à l'intérieur d'un agrégat. Donner un exemple d'invariant global à l'application (qui n'est pas localisé dans un seul agrégat).
6. Comment votre design permet-il d'annuler un contrat ?

7. Considérons la règle métier suivante : « Un contrat est valide s'il a été signé et payé ». Comment votre design prend-il en compte cette règle métier ?
8. Dans le cas où les objets sont persistés en base de données, quelles sont les deux étapes du cycle de vie d'un objet où une factory peut être utile ?
9. Lorsqu'un objet doit être persisté en base de données, il est nécessaire d'implémenter la sérialisation de cet objet. Où est-il envisageable de placer ce code ? (plusieurs possibilités) Quel endroit vous semble le plus judicieux ?