Hibernate

Hibernate = retirer toutes les lignes de plomberie de la DAO  
  
1) config > lib = tous les jar

2) hibernate.cf.xml

3) xxx(classe métier).hbn.xml

4) API Hibernate (Session, SessionFactory,…)

Question d’exam :   
- Peut-on avoir 2 sessions factory ? Oui ! chaqu’une discutant avec une base différente  
- SessionFactory connait la base, crée la base, les tables etc… Une session est un canal de discussion avec la base de données  
- Mode par défaut d’hibernate : commit intelligent/ commit / always / ?  
- différence entre persist et save ? Save rend un id, persist rend rien.  
- Un objet détaché est un objet qui a eu sa représentation en base.  
- Session = cache de 1ier niveau (quand je veux un objet, d’abord ça regarde dans la session, sinon en base)  
- Merge est couteux (comparaison de l’objet),   
- Quelle est la différence entre un session.load et get ? Get cherche dans session, puis base sinon null. Load donne toujours un objet, si on effectue un getXXX, soit il l’a en base, et c’est OK sinon exception. Si on fait un load, c’est pour des raisons d’optimisation : Exemple, lorsque l’on utilise juste l’ID. Utiliser load lorsqu’on est sur qu’il est dans la session.  
- Peut-on modifier le fichier de config programmaticalement ? Oui !  
- Attributs de classe pour mapping : A connaitre !  
- Pourquoi prendre Set au lieu de List ? List accepte redondance, Set non ! A chaque ajout, ArrayList double (lourd), Vector \* 1.5.  
- Vérifier association one-to-many. « inverse=true » se met du côté du propriétaire de la relation one-to-many. Exemple : Contact ayant 3 entreprise : inverse=true donne insert contact, insert 3e / insert 3e, 1 contact, 3e update  
- Le choix entre linked, list, set, list/linked efficace dans recherche, set pour suppr/ajout/…  
- constrained= true : s’assurer qu’une adresse à un id dans sa partie contact  
  
Pour insérer un objet :  
Session session = HibernateUtil.getSessionFactory().getCurrentSession() ;  
session.beginTransaction()  
session.save(contact) ;  
session. ????  
session.commit() ;  
  
  
Life Cycle :  
Objet uniquement crée avec ‘new’ -> Objet état transitoire  
Objet transitoire save/persist/saveOrUpdate -> Objet état persistant  
Objet persistant clear(vide la session)/evict(sors de la session)/close(ferme la session) -> Objet détaché  
Objet persistant delete -> objet transitoire (si pas de référence, alors GC)  
  
LazyLoudingException = Exception soulevé car on essaye d’accéder à un élément d’un Objet (une association) alors que la session est close  
LasyLouding : Hibernate met un proxy sur chaque association d’un Objet, dans le but de pas tout charger.

Fichier de config :  
- hbm2ddl.auto à create, sauf quelques jours avant la soutenance, passer à update  
- les données dans les log (show\_sql) sont cachées. Voir sur google jar d’affichage log

Fichier de mapping :  
- Eviter de générer automatiquement !  
- package à mettre  
- cascade pour ne faire d’un seul persist en Java (évite d’avoir un ordre). Provoque le persist de toutes les associations. Penser à construire le graphe d’Objet en mémoire : 1 Contact à plusieurs PhoneNumber  
- …Tuplelizer = Annule la dernière action faite, erreur dans le fichier de Mapping.  
  
Règles d’écriture :  
- Generer getter/setters/constructor avec et sans fields  
- Toujours utiliser un ID ! Et « idContact », « idPhone », etc …  
- Pas de classes final  
- id increment pour les TP  
- Si vous souhaitez qu’un id d’une table soit le même que celui d’une autre table (one-to-one) alors foreign