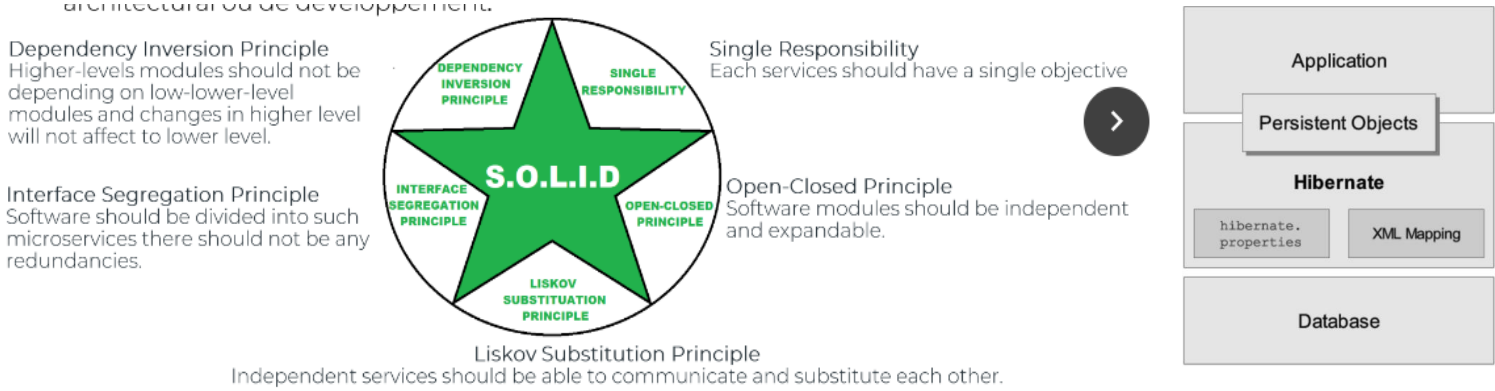
- L'architecture logicielle définit la structure d'un système, son comportement et les relations entre ses composants

- **Types d'architectures logicielles :** modèle en couches (couche présentation, métier, d'application, de données), modèle client-serveur, modèle évènementiel, modèle micro-noyau, modèle des microservices
- **Les applications d'entreprise**s permettent : la réduction des temps et coûts de développement, qualité du code, portables, adaptables, sures, sécurisée, intégrable, extensibles, maintenable
- **Les besoins exprimés** : Besoins de normalisation, Besoins d'abstraction, Besoins de communication, Besoins de composants



- Principes S.O.L.I.D : principe d'ouverture fermeture, principe de substitution de Liskov (les préconditions ne peuvent être plus fortes dans une sous-classe et les postconditions ne …. …. Plus faibles ……), une seule responsabilité, mais comment les respecter ? inversion de contrôle, injection de dépendances, programmation par aspects, design patterns
- Types d'architectures de S.I : architecture fonctionnelle, applicative, logique (arch. Applicative logique), technique
- JEE est une arch. Multi-tier qui offre le high cohesion (ensemble pour chose particulière) et low coupling (faible dépendance)
- Client léger est un S.I qui fonctionne sur un env. distant basé sur un serveur (navigateur). C. lourd est un S. qui se connecte au serveur sans avoir de réseau (microsoft outlook)
- Types de composants : composant logiciel (paquets, biblio…), d'architecture (un processus), et métier (EJB)
- Architecture JEE est défini par : composant logi. (cp. Web: JSP, SERVLET, cp. Mét. : EJB), serveur JEE (serveur de scripting ; php3, orientée objets : jsp/servlets, orientées métiers : EJB) , protocoles. Les services d'infrastructure (JDBC, JTA ,JCA) et les services de commu. (JAAS, RMI, JMS). Frameworks (Hibernate, struts, spring)
- Httpservlet contient les méthodes doPost, doGet, service (appelé lors de l'invocation de la servlet), init, destroy
- HttpServletRequest possède les méthodes : getMethod, getServerName, getParameter, getParameterNames
- ServletContext est un objet crée par le conteneur WEB et permet d'obtenir des infos de configuration du fichier web.xml, et contient les methodes : getInitParameter, getInitParameterNames, setAttribute, getAttribute, removeAttribute
- ServletConfig contient les methodes : getInitParameter, getInitParameterNames, getServletName, getServletContext
- Les paramètres sont en lecture seule et juste des objets string en contraire des attributs.
- Xml : servlet > servlet-name > servlet-class > /servlet > servlet-mapping > servlet-name > url-pattern > /servlet-mapp..
- HttpServletRespone contient les méthodes : setStatus, setContentType, getWriter, sendRedirect, forward (le traitement se fait en interne dans le même serveur pas comme send redirect), include. Ex; request.getRequestDispatcher(''/test.jsp'').forward(request,response) //// response.sendRedirect(''test.html'')
- Le cycle de vie de JSP est: jspInit() > jspService() > jspDestroy(). Les éléments d'une jsp sont : scriptlets & elmts implicites (request, out…). Ces pages sont formées par les elts : les expressions (<%= … %>), les declarations (<%! … %>), directives (<%@page [language= ''|import=''|errorPage=''|contentType=''] %>, <%@ include file=''%>), scriptlets (<% … %>), actions (<jsp:tagName[include | forward page=''] [param name='' value=''] />), JSTL (<%@taglib uri='' prefix='c' %>, <c:[out| forEach| forTokens| param| choose…when…otherwise²| redirect| if| import] > </c:if[…]>)
- HttpSession a les methodes: getSession, getSession(boolean), getAttributeNames, getAttribute, setAttribute
- Exemple MVC: model (créer student class et studentDataUtil class), view (page.jsp), controller (servlet.java)
- un pool de conn est un caches de connx de bdd maintenu afin de réutiliser les connx des futures demandes à la bdd
- → 1- créer la base de données sur mySql -> configurer tomcat database conn pool (telecharger jdbc driver .jar -> definir conn pool n webContent/META-INF/context.xml -> obtenir la réf du pool de con dans le code java)

- **PARTIE2**
- Persistence de données peut se faire : framework op.sou (HIBERNATE [la mise en œuvre de JPA]), fram. commerciaux (Toplink), API standards(JPA [pont entre le modele OOP et relationnelle], JDO, EJB entity [des compo. Serveurs qui fournissent des services à l'aide des EJB de session ou EJB entités])
- Hibernate necessite : une classe java bean, un fichier de correspondance table et classe, des propriétés de conf. Ex: connx bdd → le mapping se fait soit par fichier xml ou par annotation JPA

- Les dialectes de hibernate sont les implémentations des diff types de bdd. Le mapping se fait en : instancier la classe org.hibernate.cfg.Configuration à l'aide de fichier hibernate.cfg.xml -> association table classe à l'aide de Nmclasse.hmb.xml ou annotations jpa
- 3 états d'une classe persistence (classe implémentant les services métiers) : Passager (transient) (n'est pas encore associé pas de get, load ou save), persistant (possède une valeur clé prim.), détaché (objet était déjà associé à un contexte de persistance)
- Pour créer une classe persistante : constructeeur sans params, créer une proprie. Identifiant, classe non final, des accesseurs
- Pour utiliser hibernate : créer innstance de la classe, et la classe SesionFactory, et session (qui utilisera les services hiber.)
- **Gestion sessions** : créer; configuration config=new Configuration()->SessionFactory sesfac; -> sesfac=config.configure(''hibernate.cfg.xml'').buildSessionFactory() / ouvrir; Session sess = sesfac.openSession() /fermer : session.close()
- **Gestion des transactions** : sess.beginTransaction() -> sess.getTransation -> sess.getTransation().[commmit() | rollback()]
- Meth de sauvegarde : session.save(p) / session.persist(p) / session.update()/ session.flush(), meth de suppression : session.delete(p) / session.clear(p), récuper des objets : p= (classePersistance) session.get(classePersistance.class, clé) ou load (au lieu de get), meths de synchro: flush() (session ->bd)/ refresh() (bd->session)/ isDirty()/ evict(), requetes : SQLQuery sql = session.createSQLQuery(''req'') -> sql.ExecuteUpdate()
- **Le fichir de configuration** contient: hibernate.configuration.[driver_class | url | username | password…], hibernate.dialect
- <property name=''hbm2ddl.auto''> ??</property>: validate, update, create, create-drop, none
- hibernate-mapping -> class name='' table '' -> id name='' type='' -> <generator class='' /> -> <property name='' column='' type='' nit-null='' length=''/> → </class> → </hib…>
- main: Session s =HibernateUtil.getSessionFactory().getCurrentSession(); /s.beginTransction() /produit p= new Produit("hh", 200, "iop"); / s.save(); / s.getTransaction().commit()
- **HQL**: Query query= session.createQuery('select pers.nom from Personne as pers where pers.nom="Dupont"'); / List list= query.list() /////// Query query= session.createQuery('from Produit where pu= :x'); / query.setInteger("x", 200); / List…
- **Maven**: facilite le processus de construction, fournit un sys. De construction uniforme, et des infos utiles sur le projet
- **Annotations JPA**: @table, @column, @id, @GeneratedValue, @Transient, @OnetoMany, @JoinedColumn, @ManyTOMany
- **EntityManagerFactory**: lit le fichier persistence.xml et crée les tables à l'aide de la propriéété update, contient les methodes : persist(), find, createQuery, remove, merge. Pour le créer : private EntityManager entityManager ; / EntityManagerFactory ent= Persistence.createEntityManagerFactory(''MavenProj1'') ; / entityManager= ent. createEntityManager() ;
- **Gestion des transactions** : à l'aide de EntityTransaction (entityManager.getTransaction) et les methodes : begin, commit, rollback
- **Gestion des produit** : ajouter : entityManager.persist(p), consulter les produits : entityManager.createQuery(' ').getResultList() / ou Poduit p=entityManager.find(Produit.class, id), modifier : entityManager.merge(p), suppr : 1- etape de find ensuite -> entityManager.remove(p)
- **Mapping avec héritage** : @inheritance(strategy=InheritanceType.[SINGLE_TABLE | table per classe | joined]) -> @discriminator(name= ?, discriminatorType= ?, length=55 -> @discriminatorValue(''PF'')

Pour les relations onetomany et manytoone On utilise l'annotation:

- @ManyToOne @JoinColumn(name="ColomnName")
- @OneToMany (mappedBy="Attribut de la classe associée avec ManyToOne", fetch=FetchType.*LAZY)*
- FetchType.LAZY : indique que la relation doit être chargée à la demande
- FetchType.EAGER : indique que la relation doit être chargée en même temps que l'entité qui la porte

```
@ManyToMany
@JoinTable( name = "Magasin_Produit",    joinColumns
@JoinColumn(name = "idm"),     inverseJoinColumns = @JoinColumn(
  name = "REF" ))
```

Un projet basé sur la technologie JEE (Java Enterprise Edition) peut avoir une architecture technique complexe, mais généralement, il se compose de plusieurs couches : Couche de présentation : Cette couche gère l'interface utilisateur et les interactions avec l'utilisateur final. Elle peut utiliser des technologies telles que JavaServer Faces (JSF), JavaServer Pages (JSP) ou des frameworks de développement d'application web tels que Spring MVC..

Couche de service : Cette couche gère les fonctionnalités métier de l'application. Elle peut utiliser des technologies telles que les EJB (Enterprise Java Beans) pour implémenter des fonctionnalités de haut niveau telles que les transactions, la gestion des données et la sécurité. Couche de données : Cette couche gère l'accès aux données de l'application, en utilisant des technologies telles que JPA (Java Persistence API) pour accéder à une base de données relationnelle ou JPA-JAXB (Java Architecture for XML Binding) pour accéder à des données XML. Couche d'infrastructure : Cette couche gère les composants de base de l'application, tels que la gestion des transactions, la gestion de la sécurité, la gestion de la mémoire, etc. Elle peut utiliser des technologies telles que Java Servlets, JavaServer Pages (JSP) et JavaServer Faces (JSF) pour gérer les requêtes HTTP et les réponses.