Java Enterprise Edition est un framework. Une plate-forme fortement orientée serveur pour le développement et l'exécution d'applications distribuées. Deux parties essentielles: - Un ensemble de spécifications pour une infrastructure dans laquelle s'exécute les composants écrits en Java : serveur d'application. -Un ensemble d'APIs qui peuvent être obtenues et utilisées séparément. Framework : composants logiciels structurels pour créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel.
Java EE : Serveurs ; **Java SE** : Serveurs, PC ; **Java ME** :PDA, TV set, Mobiles, Capteurs ; **Java Card** : Cartes à puce
World wide web : langage, modèle de communication client-serveur, protocole http
Pages statiques : pages HTML préparées à l'avance. Serveur renvoi les pages sans effectuer de traitement particulier
Pages dynamiques : pages HTML générées par serveur. Serveur construit la réponse en fonction de la requête de l'utilisateur
Serveur : ordinateur disposant d'un certain nombre de ressources qu'il met à disposition d'autres ordinateurs (clients) via le réseau, ex : serveur web, d'application.
Serveur web : Protocole HTTP: répond aux requêtes des clients (navigateur web) et les traite.

	Développement	
	Côté Client	Côté Serveur
.class autonome	Applet	Servlet
Souce java mixé avec code html	Javascript (pages statiques)	Jsp (pages dynamiques)

Limites d'un serveur web seul : pas de contenu dynamique, pas de sauvegarde de données sur le serveur.
Le serveur Web a besoin d'aide pour faire du dynamique : conteneur Java EE avec servlets (ex : Tomcat)
API : est une interface de programmation. Un ensemble de fonctions, procédures ou classes mises à disposition des programmes informatiques par une bibliothèque logicielle, un système d'exploitation ou un service. **Composants** : permet un découpage de l'application et donc une séparation des rôles lors du développement : -composants web : servlet et JSP(java server pages) ; - composants métier : EJB(enterprise Java Beans). **Services** : services d'infrastructures : JDBC, JNDI, JTA, JCA, JMX ; services de communication : RMI-IIOP, JavaMail, JAAS
L'architecture d'une application se découpe idéalement en au moins trois tiers :- partie cliente : permet le dialogue avec l'utilisateur. Elle peut être composée :**application stand-alone* ; **application web* ; **applets*. – partie métier : encapsule les traitements (dans les EJB ou JavaBeans). – partie données : stocke les données
Pour une **application web dynamique, 2 grands type de pages JEE** : - Servlets : classes java spécifiques exécutées ou non sur un serveur JEE. Méthode principale appelée à chaque requête du client et recevra en paramètre la requête soumise. Après traitement (dans le corps de la méthode), elle renverra ensuite au client la page HTML générée. – JSP : même but que les Servlets mais avec une syntaxe plus proche de l'HTML
Conteneur : - assurent la gestion du cycle de vie des composants qui s'exécutent en eux. Les conteneurs fournissent des services qui peuvent être utilisés par les applications lors de leur exécution. Plusieurs conteneurs définit par JEE : conteneur web (exécuter servlets et jsp), conteneur d'EJB (exécuter les EJB), conteneur client (exécuter des applications stand-alone sur les postes qui utilisent des composants JEE)

- composant logiciel système qui contrôle d'autres composants, dits métier : ex Tomcat. Les servlets n'ont pas de méthodé main() controlés par Tomcat

Un servlet ne peut pas être déployé directement dans un conteneur, il doit faire partie d'un module Web. **Module Web** : ensemble de librairies, de fichiers de configurations, de code Java (bytecode des servlets...). C'est l'unité de déploiement dans le conteneur
Pour déployer application dans un conteneur, **2 éléments** : - application avec tous les composants (classes compilées, ressources ...) regroupée dans une archive ou module. Chaque conteneur possède son propre format d'archive. - Un fichier descripteur de déploiement contenu dans le module qui précise au conteneur des options pour exécuter l'application

Archive/module	Contenu	Extension	Descripteur de déploiement
bibliothèque	Regroupe des classes	jar	
application client	Regroupe les ressources nécessaires à leur exécution (classes, bibliothèques, images, ...)	jar	application-client.jar
web	Regroupe les servlets et les JSP ainsi que les ressources nécessaires à leur exécution (classes, bibliothèques de balises, images, ...)	War	Web.xml
EJB	Regroupe les EJB et leurs composants (classes)	jar	

Serveurs d'applications peuvent fournir : - conteneur web uniquement ; - conteneur d'EJB uniquement ; - 2 conteneurs
Servlet : spécificités : résidante sur le réseau (un mini serveur), exécuté par le moteur de Servlet, construit dynamiquement du code. Avantages : accès à tout l'API java et +, portabilité, gestion de cache, des threads
Récupérer informations envoyés par le client : request.getParameter(« monParam »)
Envoyer une réponse au client : PrintWrite out = res.getWriter() ; out.print(« <html><head>... ») ;
//ajouter code servlet, web.xml etc...
Champ manquant : getParameter => null ; Champ vide : getParameter => «»
Les servlets facilitent le traitement avec java des requêtes et réponses HTTP, mais ils ne sont pas appropriés à l'écriture de code HTML. Difficile de séparer des différentes tâches du développement.
Balises : Génération d'un affichage<%= maVarAAfficher %> ; Autoriser le multithreading <%@ isThreadSafe=« true » %> Importation de package java <%@ page import=« java.util.Calendar » %> ; Définir une page d'erreur <%@ errorPage=«error.jsp » %> Definition de la nature de la page générée <%@ page contentType=« text/html » %>;Déterminer une page comme page d'erreur <%@ isErrorPage=«true» %> Inclusion de fichier JSP <%@ include file=« file.jsp » %> Définir des variables d'instance <%! int compteur=0; %> Commentaires JSP<%-- Commentaires --%>

Déclaration de variables de session : session.setAttribute(String name, Object value) ; **Lecture d'une variable de session** : Object session.getAttribute(String name); **Déclaration de variables d'application** : application.setAttribute(String name, Object value) **Lecture d'une variable d'application** : Object application.getAttribute(String name) ; **Génération de l'affichage** : <%=..... %>
Utilisation des sciptlets pour créer des parties conditionnelles : Les scriptlets sont insérés tels quels dans le servlet
Déclarations jsp : <% ! Java Code %> <% ! private int someField =5 ;%>
Limiter le code java dans les jsp : ecrire les lignes dans une classe java et 1 ligne dans une jsp pour l'invoquer. **Pk ?** => Développement : ecriture de la classe dans un environnement java et pas html. Debogage : plus facile de voir les erreurs. Test : test + facile. Réutilisation : réutilisation de la même classe dans différentes JSP.
JavaBeans : classes Java, syntaxe : <jsp:useBean id= " nomInstanceJavaBean " class="nomClasseDuBean" scope=" equest|session|application|page"></jsp:useBean>. Constructeur vide (soit en ne mettant pas de constructeur). Pas d'attributs publics. Getters et setters
Scope : portée du bean. Positionner les propriétés du bean dans une JSP : <%= nomBean.setCompter()%> ou <jsp:setProperty name= "nomBean" property="compter" value= "6" />
MVC : modèle : Bean. Vues : JSP. Contrôleur : servlets
Syntaxe pour lancer la JSP : ServletContext context = getServletContext(); Dispatcher dispatcher = context.getRequestDispatcher("/maPageMiseEnForme.jsp"); dispatcher.forward(request, response);
La servlet peut passer des valeurs à la JSP appelé grâce à setAttribute().La JSP extrait les objets de request grâce à getAttribute()<% ArrayList theList = (ArrayList)request.getAttribute(« nomDelObjet");%>
Expression language (EL) : ${expression} La chaîne expression correspond à l'expression à interpréter. Une expression peut être composée de plusieurs termes séparés par des opérateurs. ${ (10 + 2)*2 } => 24 ou ${bean.property}
La **JSTL** permet de développer des pages JSP en utilisant des balises XML, donc avec une syntaxe proche des langages utilisés par les web designers, et leur permet donc de concevoir des pages dynamiques complexes sans connaissances du langage Java. Pour l'utiliser il faut copier dans WeB-INF/lib jstl.jar et standard.jar et dans l'entête <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>. Gestion des variables de **scope** : <c:out value="Bonjour"/> : Afficher une expression (⇔ <%= ... %>). <c:set var="maVariable" value="valeur" scope="request"/> : **Définir une variable de scope ou une propriété d'un attribut ou d'un bean.** <c:remove var="varName" scope="session"/> : **Supprimer une variable de scope.** <c:catch/> : **Intercepter les exceptions qui sont levées lors de l'exécution du code inclus dans son corps.** <c:if test="${param.select == 'choix1'}"> : **Traitement conditionnel.** <c:choose ><c :when test=</c:choose > : **Traiter différents cas mutuellement exclusifs.** <c:forEach var="i" begin="1" end="10" step ="1"> ou <c:forEach var="customer" items="${customers}"> : **Permet d'effectuer simplement des itérations sur plusieurs types de collection de données.** <c:forTokens var="mot" delims=";" items=str> : **Permet de découper des chaînes de caractères selon un ou plusieurs délimiteurs.** Chaque marqueur ainsi obtenu sera traité dans une boucle de l'itération. <c:url value="/MaPage.jsp?nom=Jean"> : **Permet de créer des URLs absolues, relatives au contexte, ou relatives à un autre contexte.** <c :redirect url= «/error.jsp »> **envoi une commande de redirection http au client.** < c :import url= « /file.jsp »> **importer une ressource selon son url.**
Hibernate : lier l'orienté objet et la base de données relationnelle à travers le mapping objet/relationnel(Object-Relational Mapping ou ORM)
JDBC permet aux applications Java d'accéder par le biais d'une interface commune à des sources de données. Pour lier notre programme à la BDD, nous devons utiliser un Driver spécifique à cette BDD. Hibernate se chargera automatiquement de la connexion à la BDD, il suffira de la lui paramétrer et de lui fournir le Driver correspondant.
Rendre **persistant** un objet c'est sauvegarder ses données sur un support non volatile de telle sorte qu'un objet identique à cet objet pourra être recréé lors d'une session ultérieure. La persistance des données des objets peut se faire via une base de données relationnelle. La gestion de la persistance des objets peut parfois être complexe : D'où l'utilité d'utiliser un framework se chargeant d'une partie de la gestion de la persistance.
Hibernate : 2 types de fichiers pour la configuration : hibernate.properties ou hibernate.cfg.xml : lier Hibernate à notre BDD et au Driver à utiliser. Nom_classe.hbm : Les fichiers de mapping, qui permettent de lier la BDD avec les objets persistants. Ils respectent la syntaxe XML. Au démarrage, Hibernate crée un objet SessionFactory. Une **SessionFactory** peut ouvrir des nouvelles Sessions. Le langage que nous utilisons pour faire des requêtes à la BDD est propriétaire à Hibernate et se nomme le HQL.
//hibernate.cfg.xml
Spring : Framework libre pour construire et définir l'infrastructure d'une application JEE, facilite le développement et les tests, conteneur léger : Le **gros avantage** par rapport aux serveurs d'application est qu'avec Spring, vos classes n'ont pas besoin d'implémenter une quelconque interface pour être prises en charge par le framework. C'est spring qui se charge d'appeler les différents composants, en fonction d'un fichier de configuration, il utilise le pattenr inversion of control (IoC). Dispose d'une couche d'abstraction, lui permettant d'intégrer d'autres frameworks et librairies très simplement, ne se positionne pas particulièrement au niveau de la couche MVC. Composé de différents **modules** : Spring AOP (programmation par aspects), Spring DAO (accès aux données, avec un support pour JDBC ,Hibernate,...), Spring Context (application web Apache Struts, JSF,...), Spring MVC (intégration du pattern Front Controller via une DispatcherServlet) Configuration du web.xml fixe, un fichier xml à configurer en fonction des redirections : dispatchers-servlet.xml.
Lorsque l'on met par exemple « welcome.htm », c'est une redirection vers le controller d'action. Le controlleur utilisé hérite **d'AbstractController** ce qui signifie qu'il ne peut contenir qu'une **seule action**, nommée handleRequestInternal. Il est également possible de gérer **plusieurs actions** pour un même controller, on hérite alors de **MultiActionController**
La couche d'abstraction permet à Spring d'intégrer très simplement de nouveaux composants/frameworks. L'installation d'Hibernate est encore plus simple si tous les fichiers sont déjà générés. Sinon, il est possible d'utiliser HibernateTools ou MyEclipse (qui intègre HibernateTools).

```
public class Promo {

   Public  List<String> getPromo(String promo){

        List<String> promoList=new ArrayList<String>();
        if(promo.equals("ing1")){
            promoList.add("DonaldDuck");
            promoList.add("MinnieMouse");
            promoList.add("Pluto");//...
        }
        else if(promo.equals("ing2")){
            promoList.add("MickeyMouse");
            promoList.add("Daisy Duck");
            promoList.add("Goofy"); //...
        }
        else {return null;}
        Return promoList;
        } }
```

```
<!DOCTYPEhtmlPUBLIC"-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type"
        content="text/html; charset=ISO-8859-1">
        <title>AREL V6.0</title>
    </head>
    <body>
        <h1 align="center"> AREL: L'école virtuelle de l'EISTI </h1>
        <form method="GET"  action="http://localhost:8080/MVC/SelectPromo">
            Sélectionner la promo à afficher:
            <select name="promo"  size="1">
                <option>ing1</option>
                <option>ing2</option>
            </select>
            <input type="SUBMIT"/>
        </form>
    </body>
</html>
```

```
<?xmlversion="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
id="WebApp_ID" version="2.5">
    <display-name>MVC</display-name>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>
    <servlet>
        <description></description>
        <display-name>SelectPromo</display-name>
        <servlet-name>SelectPromo</servlet-name>
        <servlet-class>arel.SelectPromo</servlet-class>
    </servlet>
        <servlet-mapping>
        <servlet-name>SelectPromo</servlet-name>
        <url-pattern>/SelectPromo</url-pattern>
    </servlet-mapping>
</web-app>
```

```
<%@ page import="java.util.*" %>
<%@ page language="java"  contentType="text/html;charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPEhtmlPUBLIC"-//W3C//DTDHTML4.01Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type"
        content="text/html;charset=ISO-8859-1">
        <title>Result</title>
</head>
<body>
        <%
        List<String> promoList = (List<String>) request.getAttribute("promo");
        Iterator<String> it=promoList.iterator();
        while (it.hasNext()){
        out.print("<br/>"+it.next());
        }
        %>
</body>
</html>
```

On récupère l'attribut *promo* ajouté à la requête lors de l'étape 4

```
public class SelectPromo extends javax.servlet.http.HttpServlet
                         Implements javax.servlet.Servlet{

//...
protected void doGet(HttpServletRequest request, HttpServletResponse response)
                         throwsServletException, IOException {

    String promoName = request.getParameter("promo");
    Promo promo = new Promo();
    List<String> result = promo.getPromo(promoName);

    request.setAttribute("promo", result);    // On ajoute l'attribut promo à la
    RequestDispatcher view = request.getRequestDispatcher("result.jsp");//requête


    view.forward(request, response); // On forward la requête à la JSP
    }
}
```

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();
Etudiant e = new Etudiant(); // On crée l'objet étudiant
e.setAge(21);
e.setNom("Gibbon");
e.setPrenom("Jean");
session.save(e); // On demande à Hibernate de le sauvegarder dans la BDD
session.getTransaction().commit(); // On fait un commit
HibernateUtil.getSessionFactory().close();
```

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
<session-factory>

<!-- local connection properties -->
<property name="hibernate.connection.url">jdbc:mysql://localhost/test</property> //URL de ma BDD
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property> //Driver à utiliser
<property name="hibernate.connection.username">login</property> //Login
<property name="hibernate.connection.password">password</property> //Password

<property name="current_session_context_class">thread</property>
<!-- dialect for MySQL -->
<property name="dialect">org.hibernate.dialect.MySQLDialect</property> // Mode de discussion entre Hibernate et la BDD
                                                                        // (existe par défaut pour chaque type de BDD)
<property name="hibernate.show_sql">false</property> // Hibernate affiche ou non toutes les requêtes SQL passées à la BDD
<property name="hibernate.transaction.factory_class">org.hibernate.transaction.JDBCTransactionFactory</property>
<mapping resource="Modele/Eleve.hbm"/> // Fichier de Mapping
<mapping resource="Modele/Etudiant.hbm"/> // Fichier de Mapping
</session-factory>
</hibernate-configuration>
```

```
<beans
xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://www.springframework.org/schema/p"
xsi:schemaLocation="http://www.springframework.org/schema/beans
 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<!-- On définit un bean représentant la config hibernate -->
<bean id="sessionFactory"
  class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="configLocation"
      value="classpath:hibernate/hibernate.cfg.xml">
    </property>
</bean>

<!-- On définit un bean pour chaque classe de DAO, en injectant la
propriété contenant la fabrique de session Hibernate -->
<bean id="EtudiantDAO" class="dao.EtudiantDAO">
    <property name="sessionFactory">
        <ref bean="sessionFactory" />
    </property>
</bean>
</beans>
```

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://www.springframework.org/schema/p"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

<!-- Ici je mappe l'emplacement et le suffixe de mes vues -->

<bean id="mesVues"
class="org.springframework.web.servlet.view.InternalResourceViewResolver"
p:prefix="/views/"
p:suffix=".jsp" />

<!-- Ici je mappe les différentes actions (controlleurs) -->

<bean name="/welcome.htm" class="actions.WelcomeController"/>

</beans>
```

```
public ModelAndView checkEtudiants(HttpServletRequest request,
HttpServletResponse response)
{
// L'objet ApplicationContext peut être géré par un singleton !
(pas comme dans cet exemple)
ApplicationContext ctx = new
GenericXmlApplicationContext("applicationContext.xml");

// Utilisation de la fabrique de Beans de Spring
EtudiantDAO bean = (EtudiantDAO)ctx.getBean("EtudiantDAO");
List<Etudiant> l = bean.getListEtudiant();

return new ModelAndView("vueEtudiants","liste",l);

}
```

```
publicl class WelcomeController extends AbstractController
{
    protected ModelAndView handleRequestInternal(HttpServletRequest
request,HttpServletResponse response) throws Exception
    {
        //Passage d'un bean User dans le scope de Session :
        User u = new User();
        u.setLogin("vg");
        u.setPrenom("Vincent");
        u.setNom("Gardeux");
        u.setPassword("root");

        request.getSession().setAttribute("user", u);

        // J'appelle la classe ModelAndView pour dire à Spring vers quelle vue je veux
            forwarder la requête. On peut également rajouter des beans en scope de
            request à la volée avec ce constructeur
        return new ModelAndView("accueil");
    }
}
```

```
public class EtudiantDAO extends HibernateDaoSupport
{
    /** Les méthodes ne sont plus statiques : utilisation de la
        fabrique de Beans de Spring **/

    public void addEtudiant(Etudiant e)
    {
        Session session=
getHibernateTemplate().getSessionFactory().getCurrentSession();
        session.beginTransaction();
        session.save(e);
        session.getTransaction().commit();
    }

}
```

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<html>
    <head>
        <title>Page de Test Spring</title>

        <link rel="stylesheet" type="text/css" href="css/style.css">
    </head>

    <body>
        <p>
            Bienvenue ${user.prenom} ${user.nom} !
        </p>
        <jsp:include page="footer.jsp" />
    </body>
</html>
```

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
    <welcome-file-list>
        <welcome-file>views/index.jsp</welcome-file>
    </welcome-file-list>
    <!-- Une seule Servlet est configurée, la DispatcherServlet de Spring -->
    <servlet>
        <servlet-name>dispatchers</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <!-- Toutes les url se terminant par .htm sont mappées vers la DispatcherServlet -->
    <servlet-mapping>
        <servlet-name>dispatchers</servlet-name>
        <url-pattern>*.htm</url-pattern>
    </servlet-mapping>
</web-app>
```