Université Cadi Ayyad

École Supérieure de Technologie Essaouira

Polycopié de Programmation JEE

Préparé par :

Fatiha Bendaida

Pour la Licence Professionnelle Ingénierie des Systèmes Informatiques et Logiciels

Année universitaire : **2023/2024**

Table des matières

1 Introduction à la Programmation JEE

1.1 Définition de JEE

JEE (Java Enterprise Edition) définit un modèle pour le développement d'applications d'entreprise multi-tiers basées sur le web et faisant intervenir plusieurs composants distribués. Il s'agit d'une plateforme robuste pour la création d'applications d'entreprise complexes et évolutives.

1.2 Avantages de JEE

La plateforme JEE offre plusieurs avantages significatifs pour le développement d'applications d'entreprise :

- **Extensible** : Permet d'intégrer de nouveaux besoins et services au fur et à mesure de l'évolution de l'entreprise.
- **Disponible** : Assure une haute disponibilité et tolérance aux pannes, crucial pour les applications critiques.
- **Portable** : Indépendance vis-à-vis des plateformes matérielles et logicielles sous-jacentes grâce à la nature "Write Once, Run Anywhere" de Java.
- **Sécurisée** : Intègre des règles de contrôle d'accès et permet le détachement des opérations d'authentification, offrant une sécurité renforcée.
- **Maintenable** : Facilite la maintenance et les tests d'intégration grâce à une architecture modulaire et bien définie.
- **Scalable** : Capable de gérer un nombre croissant d'utilisateurs sans dégradation significative des performances.

1.3 Architecture n-tiers

L'architecture n-tiers est un modèle d'architecture client-serveur où une application est exécutée par plusieurs composants logiciels distincts, répartis sur différentes couches logiques ou physiques.

Exemple : Architecture 3-tiers

- **Le tier de présentation** : Gère les interfaces utilisateurs et la manière dont les données sont présentées au client (ex : pages web, applications clientes lourdes).
- **Le tier des règles de gestion (ou logique métier)** : Contient la logique métier de l'entreprise et traite les requêtes du tier de présentation en interagissant avec le tier des données.
- **Le tier des données** : Fournit les points d'accès aux données, généralement une base de données, et gère la persistance des informations.

Avantages de l'architecture n-tiers :

- Le lien entre les niveaux est défini et limité à des interfaces claires, ce qui assure la modularité.
- Les interfaces garantissent la modalité et l'indépendance technologique à chaque niveau, permettant des mises à jour ou des remplacements de composants sans affecter les autres tiers.

1.4 Historique et Évolution de Java EE

L'évolution de Java EE (anciennement J2EE) a été marquée par l'ajout constant de nouvelles spécifications et l'amélioration des fonctionnalités existantes.

- **Déc. 1999 : J2EE 1.2 (10 spécifications)** - Introduction des Servlets.
- **Sept. 2001 : J2EE 1.3 (13 spécifications)**.
- **Nov. 2003 : J2EE 1.4 (20 spécifications)** - Focus sur les Web Services.
- **Mai 2006 : Java EE 5 (23 spécifications)** - Grande simplification avec EJB 3.1, Servlet 3.0, JSP 2.2, JSF 2.0, et l'introduction des Annotations.
- **Java EE 6 (28 spécifications)** - Améliorations continues.
- **Java EE 7** - Introduction de HTML5, WebSockets, JSON-P, Batch Processing, Jcache, JAX-RS 2.0, avec un accent sur la productivité.

1.5 Technologies JEE (Liste non exhaustive)

JEE englobe un vaste ensemble de spécifications et d'API pour le développement d'applications d'entreprise. Parmi les plus importantes, on trouve :

- JSR 299 (CDI - Contexts and Dependency Injection)
- Bean Validation
- JavaMail
- Java Persistence API (JPA)
- Java Transaction API (JTA)
- Java EE Connectors Architecture (JCA)
- Enterprise JavaBeans (EJB)
- Java Message Service (JMS)
- Java EE Management
- Web Services Management
- Web Services (JAX-WS, JAX-RS, SAAJ, JAX-RPC)
- Java Authorization Contract for Containers (JACC)
- Java Authentication Service Provider Interface for Containers (JASPIC)
- Java API for XML Registries (JAXR)

2 Les Servlets

2.1 Introduction aux Servlets

Une **Servlet** est un "objet" Java (une classe dérivée de `javax.servlet.GenericServlet`) localisé "côté serveur" qui fournit un service en réponse aux sollicitations de différents clients. C'est un composant serveur spécialisé fonctionnant dans un "Web Container" Java EE.

Les servlets sont généralement de type HTTP (classe dérivée de `javax.servlet.http.HttpServlet`) et sont utilisées pour étendre des applications hébergées par un serveur web en répondant aux requêtes HTTP.

Quelques avantages des Servlets :

— **Portabilité** : Grâce à Java.

— **Puissance** : Accès à la totalité de l'API Java (Java Beans, bases de données, annuaires, web services, fonctionnalités réseau, etc.).

— **Rapidité** : La Servlet est chargée une seule fois par le conteneur, ce qui réduit le temps de démarrage pour les requêtes subséquentes.

2.2 Modèle de Fonctionnement

Le modèle de fonctionnement d'une servlet est le suivant :

1. Un **poste client** envoie une **requête** au **serveur web**.
2. Le serveur web transfère la requête au **conteneur de servlets**.
3. Le conteneur de servlets instancie la servlet (si ce n'est déjà fait) et exécute sa méthode `service()`.
4. La servlet génère une **réponse** et la renvoie au conteneur.
5. Le conteneur transmet la réponse au serveur web.
6. Le serveur web renvoie la réponse au client.

2.3 Cycle de Vie d'une Servlet

Le cycle de vie d'une servlet est géré par le conteneur web et comprend trois phases principales, chacune associée à une méthode spécifique :

— `init()` :
  — Appelée une seule fois par le conteneur, lors du premier chargement de la servlet (ou au démarrage du serveur si configuré).
  — Utilisée pour l'initialisation de la servlet (ex : chargement de ressources, initialisation de connexions).
  — Prend un objet `ServletConfig` en paramètre, qui permet d'accéder aux paramètres d'initialisation spécifiques à la servlet.

— `service(ServletRequest req, ServletResponse res)` :
  — Appelée à chaque requête du client.
  — C'est la méthode principale où la logique de traitement de la requête est implémentée.
  — Pour les servlets HTTP, cette méthode délègue généralement aux méthodes `doGet()`, `doPost()`, `doPut()`, `doDelete()`, etc., en fonction de la méthode HTTP de la requête.

— `destroy()` :
  — Appelée une seule fois par le conteneur, juste avant que la servlet ne soit déchargée de la mémoire (ex : lors de l'arrêt du serveur ou du redéploiement de l'application).
  — Utilisée pour libérer les ressources allouées par la servlet (ex : fermer les connexions à la base de données).

2.4 Méthodes `doGet()` et `doPost()`

Dans une `HttpServlet`, la méthode `service()` est généralement surchargée pour dispatcher les requêtes aux méthodes spécifiques selon la méthode HTTP utilisée par le client :

- `doGet(HttpServletRequest request, HttpServletResponse response)` :
  - Traitement des requêtes HTTP de type GET.
  - Utilisée pour récupérer des informations, afficher des pages, etc.
  - Les paramètres sont généralement passés dans l'URL.
- `doPost(HttpServletRequest request, HttpServletResponse response)` :
  - Traitement des requêtes HTTP de type POST.
  - Utilisée pour soumettre des données (ex : formulaires), créer ou modifier des ressources.
  - Les paramètres sont généralement passés dans le corps de la requête.

2.5 La Classe `HttpServletRequest`

L'objet `HttpServletRequest` représente la requête HTTP envoyée par le client au serveur. Il fournit des méthodes pour accéder aux informations de la requête :

- `getParameter(String name)` : Récupère la valeur d'un paramètre de la requête.
- `getParameterValues(String name)` : Récupère toutes les valeurs d'un paramètre (utile pour les cases à cocher multiples).
- `getMethod()` : Retourne la méthode HTTP de la requête (GET, POST, etc.).
- `getRequestURI()` : Retourne l'URI de la requête.
- `getContextPath()` : Retourne le chemin du contexte de l'application web.
- `getSession()` ou `getSession(boolean create)` : Permet d'accéder à la session HTTP associée à la requête (voir section suivante).
- `setAttribute(String name, Object value)` : Permet de stocker des attributs dans la portée de la requête, accessibles par d'autres composants de l'application (ex : JSP).
- `getAttribute(String name)` : Récupère un attribut stocké dans la portée de la requête.

2.6 La Classe `HttpServletResponse`

L'objet `HttpServletResponse` représente la réponse HTTP que la servlet envoie au client. Il offre des méthodes pour contrôler la réponse :

- `setContentType(String type)` : Définit le type MIME du contenu de la réponse (ex : "text/html", "application/json").
- `getWriter()` : Retourne un objet `PrintWriter` pour écrire des données textuelles dans le corps de la réponse.
- `getOutputStream()` : Retourne un objet `ServletOutputStream` pour écrire des données binaires dans le corps de la réponse.

— `sendRedirect(String location)` : Envoie une réponse de redirection au client vers une nouvelle URL.

— `setStatus(int sc)` : Définit le code de statut HTTP de la réponse (ex : 200 OK, 404 Not Found).

— `addCookie(Cookie cookie)` : Ajoute un cookie à la réponse.

2.7 Gestion des Sessions

La gestion des sessions permet de suivre l'état d'un utilisateur à travers plusieurs requêtes HTTP, qui sont par nature "sans état".

— **Le rôle des sessions** : Maintenir des informations spécifiques à un utilisateur pendant la durée de sa visite sur le site web.

— **Obtention/Création d'une session** :

  — `request.getSession()` : Retourne la session HTTP existante pour la requête actuelle, ou crée une nouvelle session si aucune n'existe.

  — `request.getSession(false)` : Retourne la session HTTP existante ou `null` si aucune session n'existe.

— **Ajout et récupération d'attributs de session** :

  — `session.setAttribute(String name, Object value)` : Stocke un objet dans la session.

  — `session.getAttribute(String name)` : Récupère un objet de la session.

  — `session.removeAttribute(String name)` : Supprime un objet de la session.

— **Invalidation de session** :

  — `session.invalidate()` : Invalide la session, libérant toutes les ressources associées.

2.8 Le Contexte de Servlet (`ServletContext`)

Le `ServletContext` représente le contexte de l'application web. C'est un objet unique par application web et sa durée de vie correspond à celle de l'application.

— **Rôle et durée de vie** : Il permet de partager des informations et des ressources (ex : connexions à la base de données, paramètres de configuration) entre toutes les servlets et JSP de l'application.

— **Configuration du contexte dans `web.xml`** :

```
<web-app>
  <context-param>
    <param-name>emailSupport</param-name>
    <param-value>support@example.com</param-value>
  </context-param>
</web-app>
```

— **Utilisation du contexte** :

  — `getServletContext().getInitParameter(String name)` : Récupère un paramètre d'initialisation du contexte.

— `getServletContext().setAttribute(String name, Object value)` : Permet de stocker des attributs à l'échelle de l'application.
— `getServletContext().getAttribute(String name)` : Récupère un attribut stocké à l'échelle de l'application.

2.9 La Configuration des Servlets (`web.xml`)

Le descripteur de déploiement `web.xml` (ou les annotations à partir de Servlet 3.0) est utilisé pour configurer les servlets dans une application web :

— `<servlet>` : Définit une servlet.
  — `<servlet-name>` : Nom logique de la servlet.
  — `<servlet-class>` : Nom qualifié complet de la classe de la servlet.
  — `<init-param>` : Paramètres d'initialisation spécifiques à cette servlet.

Exemple de configuration de servlet avec paramètres d'initialisation

```
    <servlet>
      <servlet-name>MyServlet</servlet-name>
      <servlet-class>com.example.MyServlet</servlet-class>
      <init-param>
        <param-name>message</param-name>
        <param-value>Hello from Servlet!</param-value>
      </init-param>
    </servlet>

```

— `<servlet-mapping>` : Associe une servlet à une URL.
  — `<servlet-name>` : Nom logique de la servlet à mapper.
  — `<url-pattern>` : Modèle d'URL auquel la servlet répondra (ex : `/my-servlet`, `/docs/*`).

Exemple de mapping de servlet

```
    <servlet-mapping>
      <servlet-name>MyServlet</servlet-name>
      <url-pattern>/hello</url-pattern>
    </servlet-mapping>

```

2.10 Les Filtres

Les filtres sont des composants qui peuvent intercepter les requêtes des clients avant qu'elles n'atteignent la ressource cible (servlet, JSP) et les réponses avant qu'elles ne soient renvoyées au client. Ils peuvent être chaînés.

— **Introduction et concept de chaînes de filtres** : Un filtre peut effectuer des opérations avant/après le traitement de la requête par la ressource cible. Si plusieurs filtres sont configurés, ils forment une chaîne et s'exécutent séquentiellement.
— **Configuration des filtres dans `web.xml`** :

— `<filter>` : Définit un filtre.

Exemple de définition de filtre

```
    <filter>
      <filter-name>logFilter</filter-name>
      <filter-class>com.example.LogFilter</filter-class>
    </filter>

```

— `<filter-mapping>` : Associe un filtre à une URL ou à une servlet.

Exemple de mapping de filtre

```
    <filter-mapping>
      <filter-name>logFilter</filter-name>
      <url-pattern>/*</url-pattern>
    </filter-mapping>
    <filter-mapping>
      <filter-name>authFilter</filter-name>
      <servlet-name>MyServlet</servlet-name>
    </filter-mapping>

```

— **Implémentation d'un filtre** : Une classe de filtre doit implémenter l'interface `javax.servlet.Filter`.
  — `init(FilterConfig filterConfig)` : Appelée une seule fois lors de l'initialisation du filtre.
  — `doFilter(ServletRequest request, ServletResponse response, FilterChain chain)` : Point d'entrée principal du filtre.
    — `chain.doFilter(request, response)` : Permet de passer la requête à la ressource suivante dans la chaîne (un autre filtre ou la ressource cible). Si cette méthode n'est pas appelée, la requête est "bloquée" par le filtre.
  — `destroy()` : Appelée une seule fois lors de la destruction du filtre.
— **Cas d'usage des filtres** :
  — Logging et traçabilité des requêtes.
  — Compression des données.
  — Authentification et autorisation (bloquer l'accès si non autorisé).
  — Modification des en-têtes de requête ou de réponse.
  — Statistiques d'utilisation.

3 Java Server Pages (JSP)

3.1 Introduction aux JSP

Une JSP (JavaServer Page) est une page HTML qui peut contenir du code Java exécutable côté serveur afin de la rendre dynamique. Les JSP sont basées sur la technologie des servlets et sont, en fait, converties en Servlets par le moteur de Servlets lors du premier appel à la JSP.

— **Objectif principal** : Elles permettent une certaine séparation entre le traitement de la requête (souvent effectué par des Servlets) et le rendu de la page (génération du flux HTML). Les JSP sont principalement destinées à la présentation (mise en forme) et non à la réalisation de traitements lourds, qui devraient être délégués à des JavaBeans ou des Servlets.

— **Accessibilité** : Les JSP, étant des documents, sont référencées par des URL (ex : `http://localhost:8080/AppliWebJSP/index.jsp`).

3.2 Séparation des Rôles

L'utilisation des JSP favorise une meilleure séparation des préoccupations dans le développement web :

— Un **développeur Java** réalise des composants (servlets, objets métiers ou techniques, etc.).

— Un **développeur de pages JSP** utilise ces composants pour les présenter au format HTML, XML, etc.

Ainsi, la logique métier est nettement séparée de la présentation des résultats, améliorant la maintenabilité et la collaboration.

3.3 Éléments de base d'une page JSP

Les JSP permettent d'insérer du code Java et de contrôler le comportement de la page à l'aide de différents éléments.

3.3.1 Directives

Les directives JSP contrôlent le comportement global de la page JSP. Elles sont traitées au moment de la traduction de la JSP en Servlet.

— `<%@ page ... %>` : Définit les attributs de la page JSP.
  — `language="java"` : Langage de script (par défaut Java).
  — `import="java.util.*, com.example.*"` : Importe des classes et packages Java.
  — `session="true|false"` : Indique si la page participe à une session HTTP (par défaut `true`).
  — `isErrorPage="true|false"` : Indique si la page est une page d'erreur.
  — `contentType="text/html;charset=UTF-8"` : Définit le type MIME et l'encodage de la réponse.
  — `buffer="8kb|none"` : Spécifie la taille du buffer de sortie.
  — `autoFlush="true|false"` : Indique si le buffer doit être vidé automatiquement.
  — `errorPage="/error.jsp"` : Chemin vers une page d'erreur en cas d'exception.

— `<%@ include file="fichier.jsp" %>` : Inclut statiquement le contenu d'un autre fichier JSP, HTML, etc., au moment de la traduction. Le contenu est fusionné avant la compilation.

— `<%@ taglib prefix="prefix" uri="URI_bibliotheque_tags" %>` : Déclare l'utilisation d'une bibliothèque de tags personnalisée ou standard (comme JSTL).

3.3.2 Scriptlets

Les scriptlets permettent d'inclure du code Java exécutable directement dans la page JSP.

— **Syntaxe** : `<% code Java %>`

— **Exemple de scriptlet JSP** :

```
<%
int anneeCourante = 2024; out.println("L'anne est : " +
anneeCourante);
%>
```

3.3.3 Expressions

Les expressions JSP permettent d'afficher la valeur d'une expression Java directement dans le flux de sortie.

— **Syntaxe** : `<%= expression %>`

— **Exemple d'expression JSP** :

```
<p>La date actuelle est : <%= new java.util.Date() %></p>
```

3.3.4 Déclarations

Les déclarations JSP permettent de définir des méthodes ou des variables de membre qui seront disponibles dans la Servlet générée à partir de la JSP.

— **Syntaxe** : `<%! code Java de déclaration %>`

— **Exemple de déclaration JSP** :

```
<%!
public String getMessage() {
  return "Bonjour du JSP !";
}
%>
<p><%= getMessage() %></p>
```

3.4 Objets Implicites JSP

Le conteneur web met à disposition des objets implicites directement utilisables dans une page JSP, sans qu'il soit nécessaire de les déclarer.

— `request` : Objet `HttpServletRequest`, représente la requête du client.

— `response` : Objet `HttpServletResponse`, représente la réponse au client.

— `session` : Objet `HttpSession`, représente la session utilisateur.

— `application` : Objet `ServletContext`, représente le contexte de l'application web.

— `pageContext` : Objet `PageContext`, donne accès aux différents scopes (page, request, session, application).

— `config` : Objet `ServletConfig`, informations de configuration de la servlet générée par la JSP.

— `out` : Objet `JspWriter`, flux de sortie pour écrire du contenu dans la réponse.

— `page` : Représente la servlet générée par la JSP (équivalent à `this`).

— `exception` : Objet `Throwable`, disponible uniquement dans les pages d'erreur (`isErrorPage="true"`).

3.5 Actions Standard JSP

Les actions standard JSP sont des tags XML prédéfinis qui permettent de contrôler le flux d'exécution et d'interagir avec les composants Java.

— `<jsp:include page="relativeURL" flush="true"/>` : Inclut dynamiquement le contenu d'une autre ressource (JSP, HTML, Servlet) au moment de l'exécution.

— `<jsp:forward page="relativeURL"/>` : Transfère la requête et la réponse vers une autre ressource (JSP, Servlet) sur le même serveur. Le client n'est pas redirigé.

— `<jsp:param name="paramName" value="paramValue"/>` : Utilisé avec `<jsp:include>` ou `<jsp:forward>` pour ajouter des paramètres à la requête.

— `<jsp:useBean id="nomBean" class="package.ClasseBean" scope="page|request|session|application"/>` :
  — Crée une instance d'un JavaBean ou référence une instance existante dans un scope donné.
  — `id` : Nom de la variable du bean.
  — `class` : Nom qualifié complet de la classe du bean.
  — `scope` : Portée du bean (par défaut `page`).

— `<jsp:setProperty name="nomBean" property="nomPropriete" value="valeur"/>` :
  — Définit la valeur d'une propriété d'un JavaBean.
  — `property="*"` : Définit toutes les propriétés dont les noms correspondent aux paramètres de la requête.
  — `param="nomParametre"` : Utilise la valeur d'un paramètre de la requête pour la propriété.

— `<jsp:getProperty name="nomBean" property="nomPropriete"/>` : Récupère la valeur d'une propriété d'un JavaBean et l'affiche.

3.6 Expression Language (EL)

L'Expression Language (EL) est un langage simple et compact utilisé dans les JSP pour accéder aux données stockées dans les scopes (page, request, session, application) et aux propriétés des JavaBeans, des Maps, etc. Il simplifie grandement l'accès aux données par rapport aux scriptlets.

— **Syntaxe** : `$${expression}`

— **Accès aux propriétés** :

- `$${bean.propriete}` : Accède à la propriété `propriete` du bean.
- `$${map["cle"]}` ou `$${map.cle}` : Accède à une valeur dans une Map.
- `$${param.nomParametre}` : Accède à un paramètre de la requête.
- `$${sessionScope.nomAttribute}` : Accède à un attribut dans la portée de session.

— **Opérateurs** :

- Arithmétiques : `+`, `-`, `*`, `/` ou `div`, `%` ou `mod`
- Logiques : `and`, `or`, `not`
- Relationnels : `==` ou `eq`, `!=` ou `ne`, `<` ou `lt`, `>` ou `gt`, `<=` ou `le`, `>=` ou `ge`
- `empty` : Vérifie si une collection, un tableau, une chaîne ou un objet est vide ou null.

— **Exemple d'utilisation d'EL** :

```
<p>Nom utilisateur : ${sessionScope.user.username}</p>
<p>Message de la requte : ${requestScope.message}</p>
<p>Total du panier : ${cart.totalPrice * 1.2}</p>
<c:if test="${empty param.name}">
  <p>Veuillez entrer votre nom.</p>
</c:if>
```

3.7 JavaServer Pages Standard Tag Library (JSTL)

JSTL est une collection de tags standardisés qui encapsulent des fonctionnalités courantes dans les JSP, réduisant ainsi le besoin de scriptlets Java.

— **Introduction et avantages** :

- Simplification et amélioration de la lisibilité du code JSP.
- Réutilisabilité des fonctionnalités courantes.
- Meilleure séparation des préoccupations (logique de présentation vs logique métier).
- Facilite la maintenance.

— **Catégories de tags** : JSTL est divisée en plusieurs bibliothèques de tags :

- **Core** (`c`) : Tags pour le contrôle de flux (if, forEach), les variables, les URL.
- **Formatting** (`fmt`) : Tags pour la mise en forme des nombres, dates, devises.
- **SQL** (`sql`) : Tags pour l'accès aux bases de données (à utiliser avec prudence dans les JSP pour des raisons de séparation).
- **XML** (`x`) : Tags pour le traitement XML.
- **Functions** (`fn`) : Fonctions d'EL pour manipuler des chaînes et des collections.

— **Exemples de tags JSTL Core (préfixe `c`)** :

- `<c:out value="expression" escapeXml="true|false"/>` : Affiche la valeur d'une expression EL.
- `<c:set var="variableName" value="expression" scope="page|request|session|app` Définit une variable dans un scope donné.

— `<c:if test="conditionEL">...</c:if>` : Exécute le corps du tag si la condition est vraie.

— `<c:forEach var="item" items="collectionEL" varStatus="statusVar"> ..... </c:forEach>` : Itère sur une collection ou un tableau. `varStatus` fournit des informations sur l'itération.

— `<c:choose> <c:when test="condition1">...</c:when>`
`<c:when test="condition2">...</c:when>`
`<c:otherwise>...</c:otherwise> </c:choose>` : Structure conditionnelle de type `if-else if-else`.

— **Exemples de Fonctions JSTL (préfixe `fn`)** : Pour utiliser les fonctions JSTL, vous devez importer la bibliothèque de tags :

```
<%@ taglib prefix="fn"
uri="http://java.sun.com/jsp/jstl/functions"%>

```

— `$${fn:length(collectionOrString)}$$` : Retourne la taille d'une collection ou la longueur d'une chaîne.

— `$${fn:contains(string, substring)}$$` : Vérifie si une chaîne contient une sous-chaîne.

Exemple d'utilisation de fonctions JSTL

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fn"
uri="http://java.sun.com/jsp/jstl/functions" %>

<c:set var="tempStr" value="Java is the best Language" />
<p>Longueur de la chane : ${fn:length(tempStr)}</p>
<p>Contient "test" ? : ${fn:contains(tempStr, "test")}</p>
<p>Contient "Java" ? : ${fn:contains(tempStr, "Java")}</p>

```