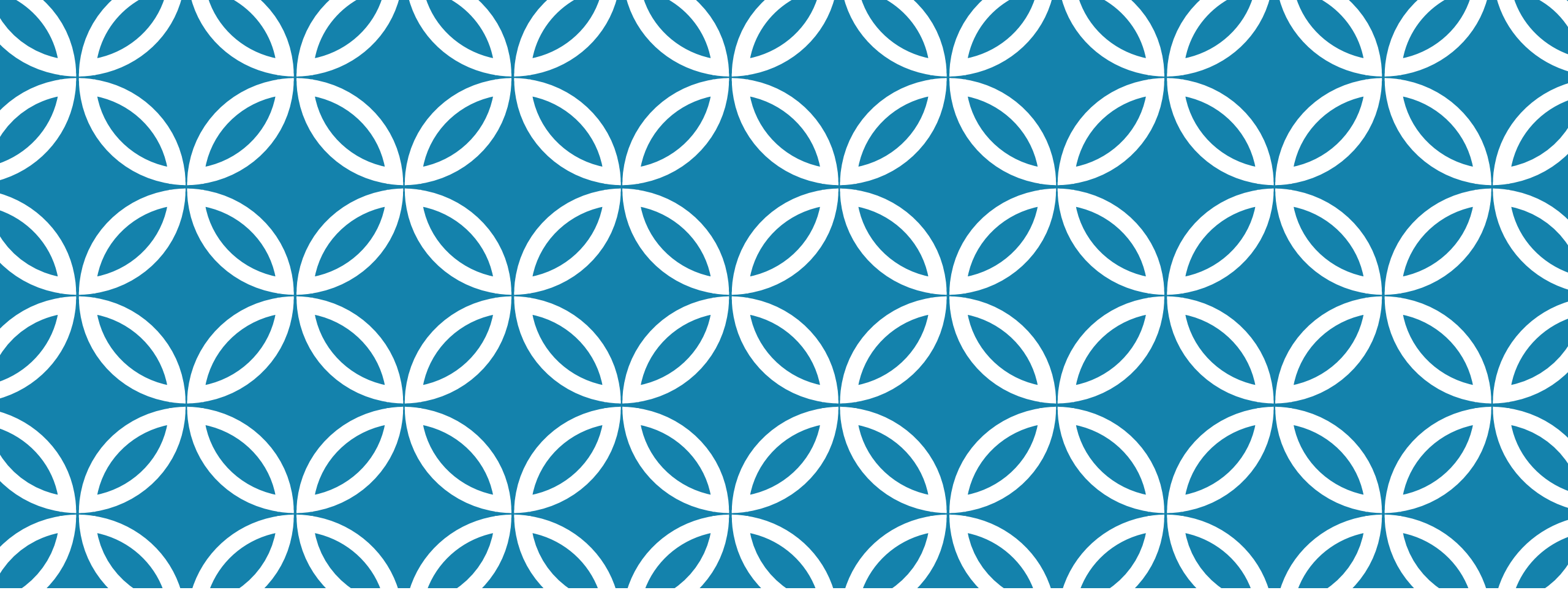
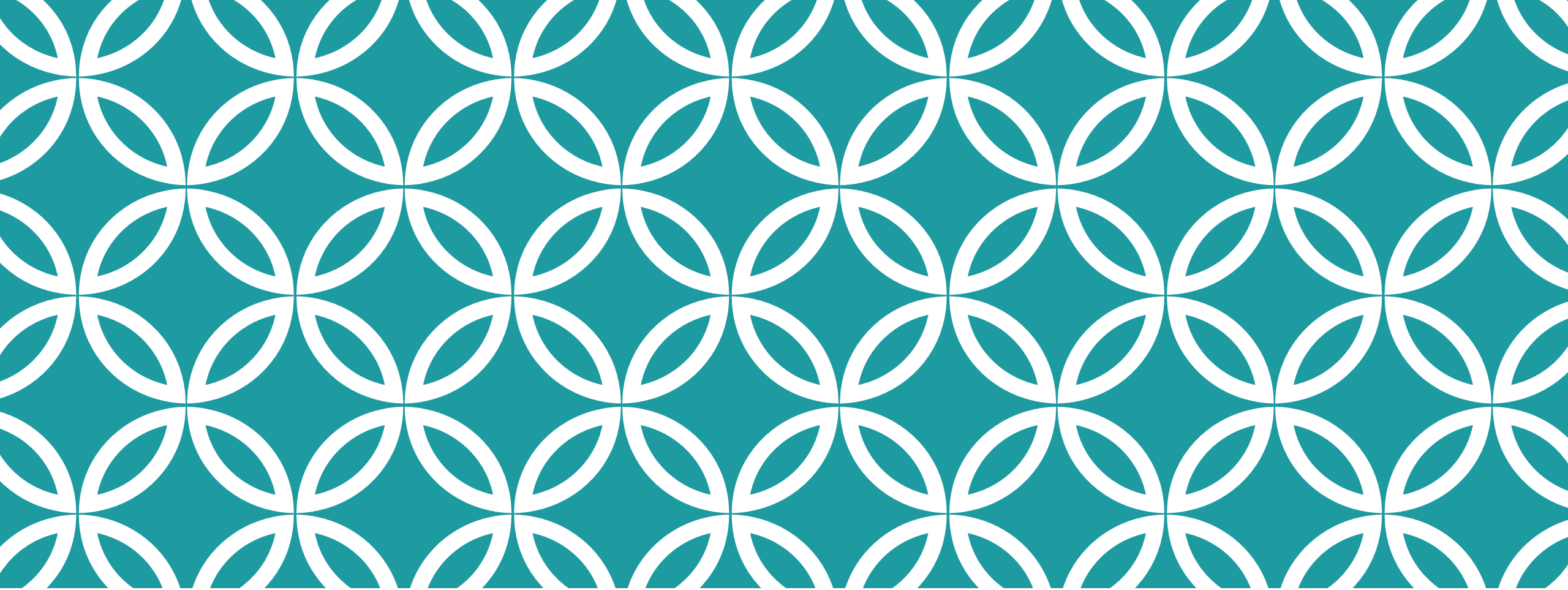


JAVA EE

Jérémy PERROUAULT



RAPPELS

HTTP & rappels

PROTOCOLE HTTP

Les commandes HTTP

Les requêtes HTTP

Les sessions HTTP

PROTOCOLE HTTP

Quelques commandes HTTP

- GET
- POST
- PUT
- DELETE

PROTOCOLE HTTP

Les requêtes HTTP composées de

- Commande HTTP, URL et version du protocole utilisé
- Champs d'en-tête [optionnels]
- Corps de la requête [optionnel]

L'URL est formée de la manière suivante

- [protocol]://[host]:[port][request_path]?[query_string]
- http://localhost:8080/formation-web?id=34
- http://localhost:8080/formation-web?idProduit=42&fournisseurId=2
- https://www.google.fr/

PROTOCOLE HTTP

La réponse HTTP est composée de :

- Statut et version du protocole utilisé
- Champs d’en-tête [optionnels]
- Corps de la réponse

PROTOCOLE HTTP

Code	Type	Message	Définition
200	Succès	OK	Requête traitée avec succès
400	Erreur (côté client)	Bad Request	La syntaxe de la requête est erronée
401		Unauthorized	Une authentification est requise
403		Forbidden	Le serveur refuse d'exécuter la requête. S'authentifier n'y changera rien.
404		Not Found	La ressource n'a pas été trouvée
405		Method Not Allowed	Méthode de requête non autorisée
408		Request Time-out	Temps d'attente d'une réponse serveur écoulé
500	Erreur (côté serveur)	Internal Server Error	Erreur interne du serveur
502		Bad Gateway	Mauvaise réponse envoyée à un serveur intermédiaire par un autre serveur
503		Service Unavailable	Service temporairement indisponible

PROTOCOLE HTTP

Le protocole HTTP est un protocole dit « déconnecté »

- Nativement, le serveur ne garde pas en mémoire l'historique des requêtes d'un client
- Il n'y a pas de cohérence client / serveur
- Exemple
  - Un client devra envoyer son nom d'utilisateur et son mot de passe à chaque requête pour que le serveur puisse le reconnaître
    - Avec toute autre information nécessaire au bon fonctionnement de l'application Web
- Pour assurer cette cohérence, l'utilisation de 2 mécanismes est primordiale
  - D'abord côté client, avec les Cookies
  - Puis côté serveur, avec les Sessions

PROTOCOLE HTTP – COOKIES

Le cookie est une donnée stockée sur le poste client (navigateur)

Envoyé à chaque requête vers l'hôte pour lequel le cookie a été stocké

PROTOCOLE HTTP – SESSIONS

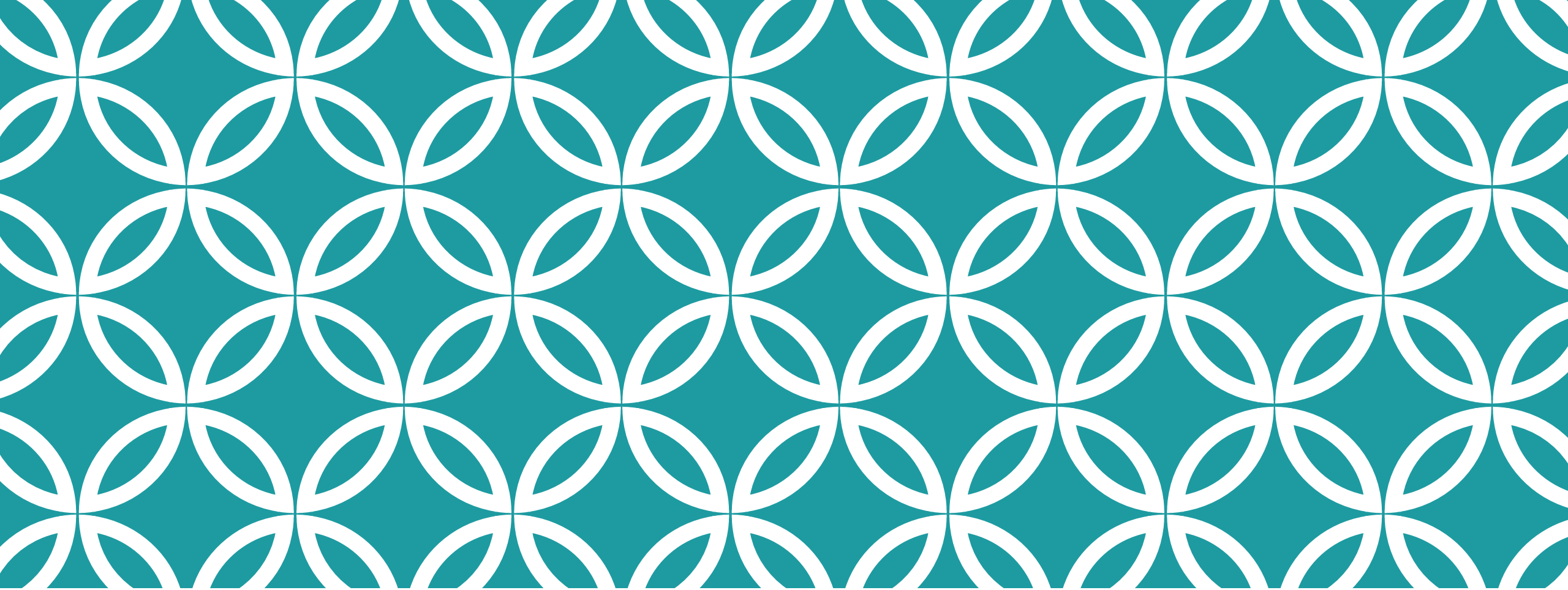
Espace alloué sur le serveur (donnée stockée sur le serveur)

Permet la persistance de données

- En Java, les informations sont stockées dans l'objet persistant **HttpSession**

Permet de maintenir la cohésion entre utilisateur et la requête

L'identifiant de session est stocké dans un cookie du navigateur !



INTRODUCTION

Introduction à JEE

INTRODUCTION

JavaEE est une norme, il faut choisir son implémentation

- Tomcat
- JBoss
- Glassfish
- …

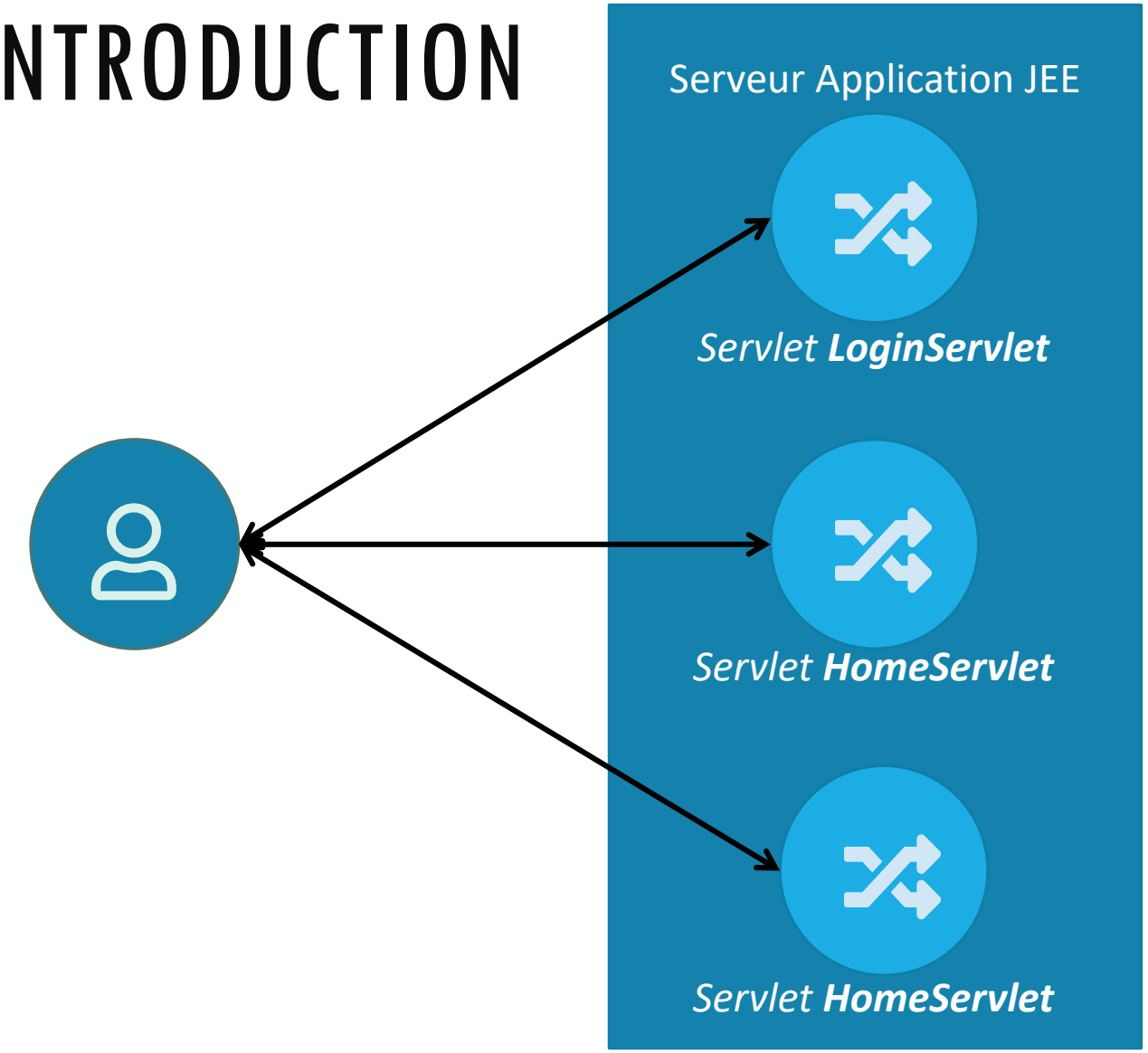
Application en couches MVC

INTRODUCTION

Le serveur d'application joue le rôle de l'application principale Java

- Il a accès à la console et peut imprimer des informations et des erreurs
- C'est un programme qui existe et a été écrit
  - On n'écrit plus de classe avec la méthode statique *main*
- C'est un programme un peu différent
  - On pourra y accéder grâce à des points d'accès (appelés *Servlet*) avec un client HTTP (navigateur web)
- Les projets web seront exécutés au sein de ce serveur d'application

INTRODUCTION



INTRODUCTION

Servlet	Listener	Filter
Point d'accès Accessible par une requête HTTP (GET, POST, PUT, DELETE, ..,) sur une URL spécifiée http://localhost:8080/projet-web/home *On appelle ça le "mapping"* C'est elle qui génère la vue et la retourne au navigateur	Ecouteur qui déclenche une action lorsqu'un évènement se déclenche	Permet de filtrer des requêtes

INTRODUCTION

Dans une application Java classique

- Un fichier JAR (Java ARchive) est créé (classes compilées et empaquetée)

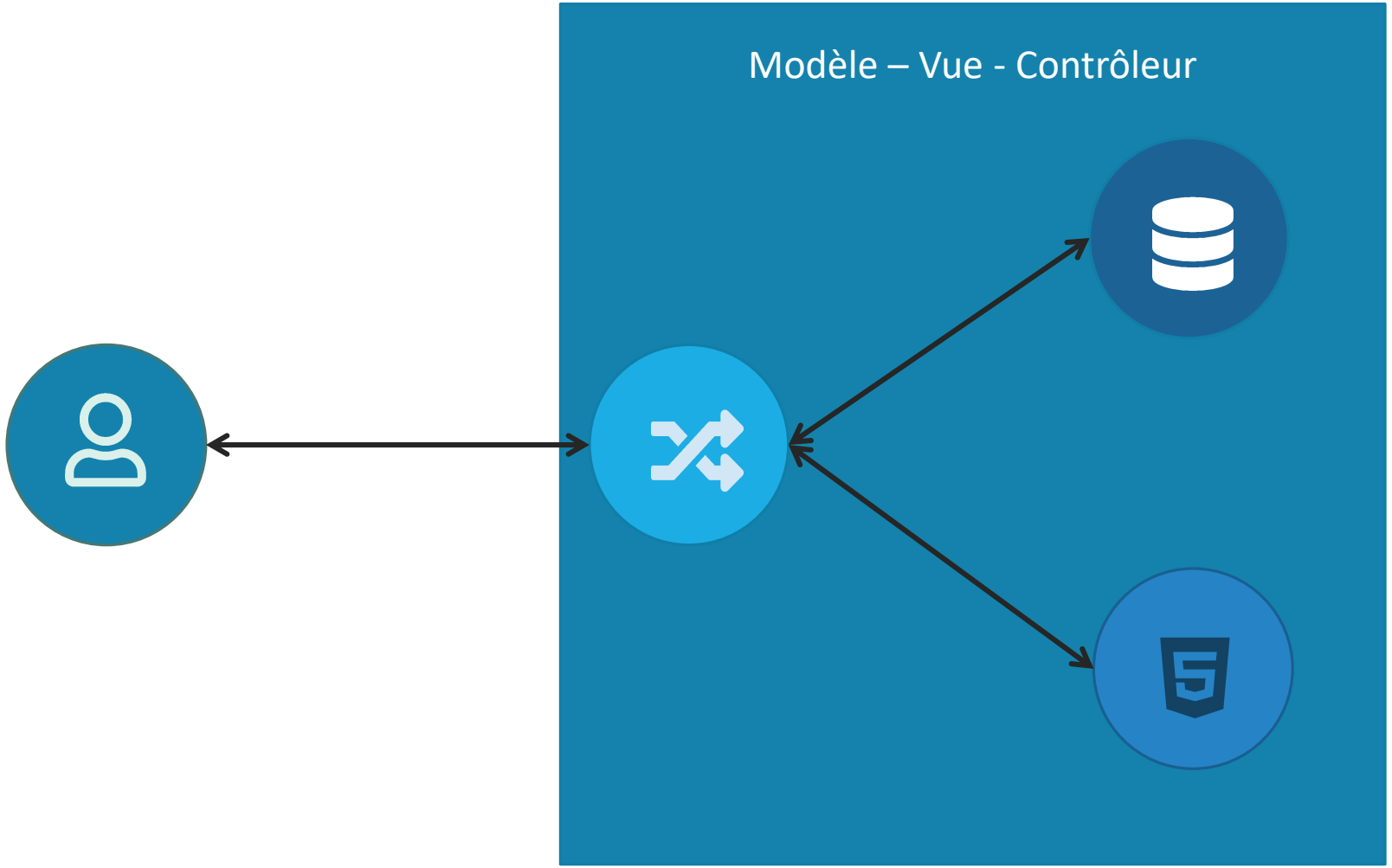
Dans une application Java web

- Un fichier WAR (Web ARchive) est créé (classes compilées et empaquetée)

Pour un projet MAVEN, il faudra donc choisir dans les options du projet

- Un package WAR

INTRODUCTION



ENVIRONNEMENT DE TRAVAIL

ENVIRONNEMENT DE TRAVAIL

Environnement Eclipse

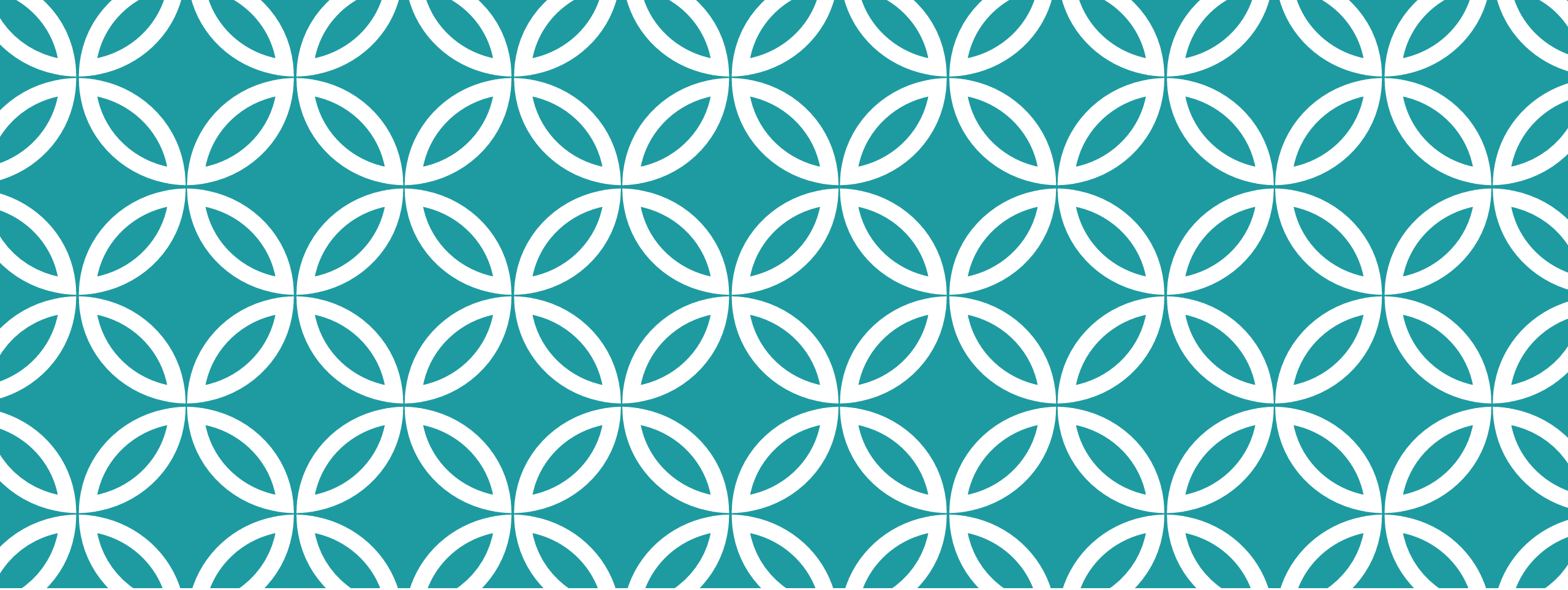
Serveur d'application Apache Tomcat

- Très léger
- Par défaut, le port utilisé par Apache Tomcat est 8080
- Le serveur web sera accessible à l'adresse http://localhost:8080/
- Chaque projet seront accessibles aux adresses http://localhost:8080/nom-projet/

ENVIRONNEMENT DE TRAVAIL – EXERCICE

Préparer l'environnement Eclipse JEE

- Perspective Java EE
- Télécharger et dézipper Apache Tomcat 9
- Configurer Apache Tomcat 9 sur Eclipse
  - Ajouter un nouveau *Server*
- Démarrer le serveur
  - Vérifier que ça fonctionne en allant sur cette adresse http://localhost:8080/
  - Une erreur 404 s'affiche si tout va bien !



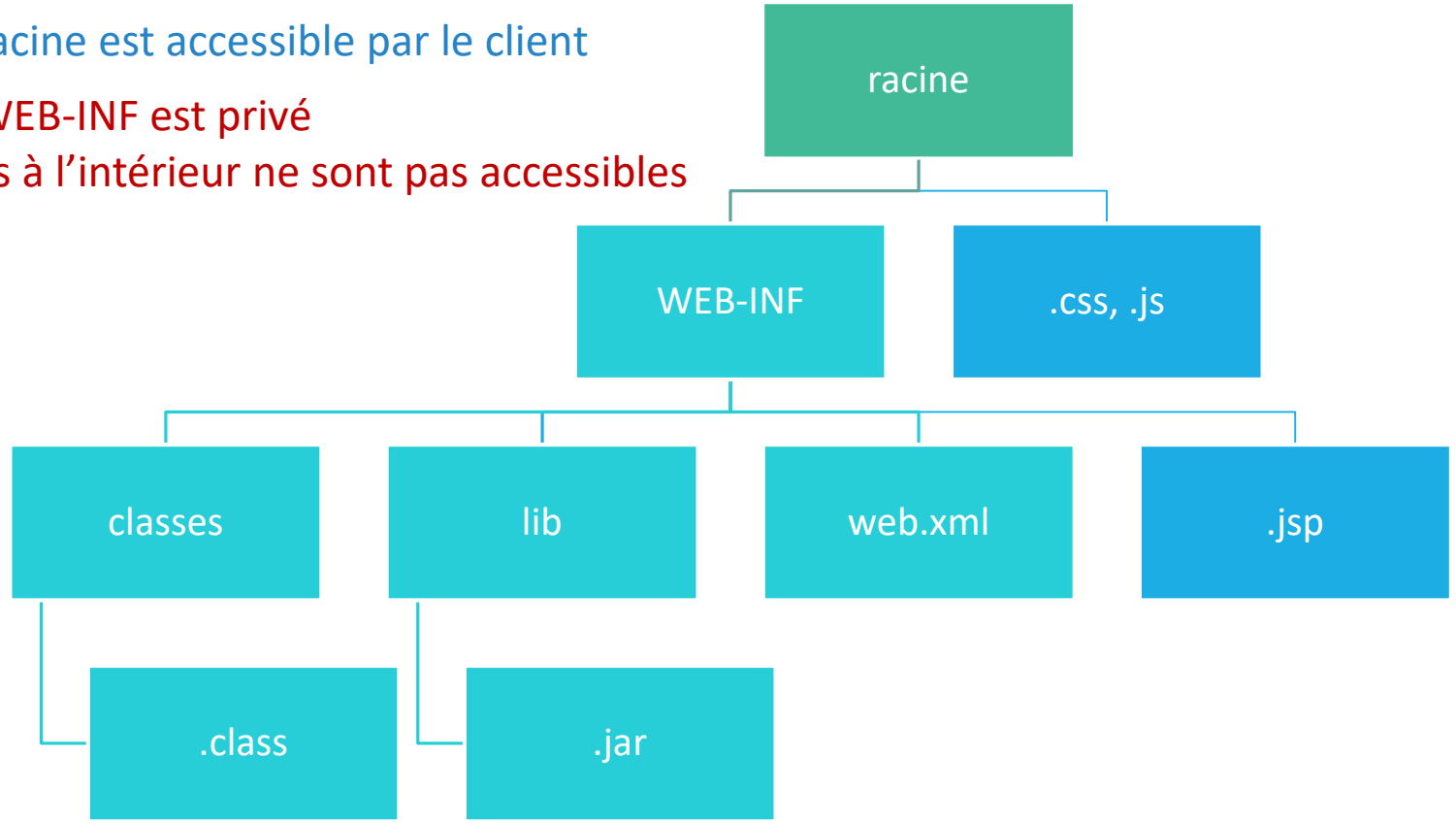
DÉVELOPPEMENT WEB EN JAVA

Les implémentations
Les servlets / JSP / JSTL
Les écouteurs & les filtres

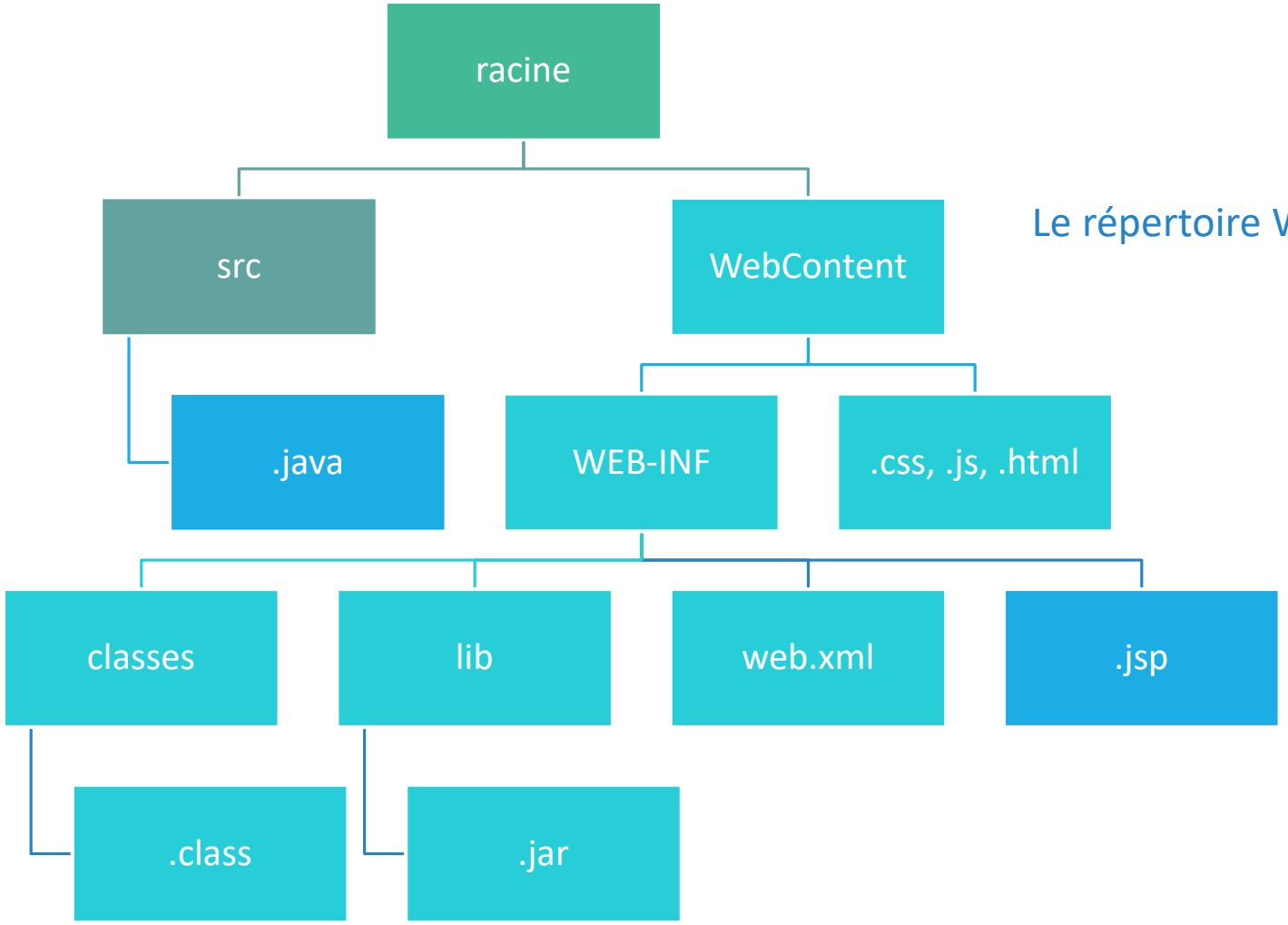
ARCHITECTURE WEB (COMPILÉ)

Le répertoire racine est accessible par le client

Le répertoire WEB-INF est privé
Tous les fichiers à l'intérieur ne sont pas accessibles

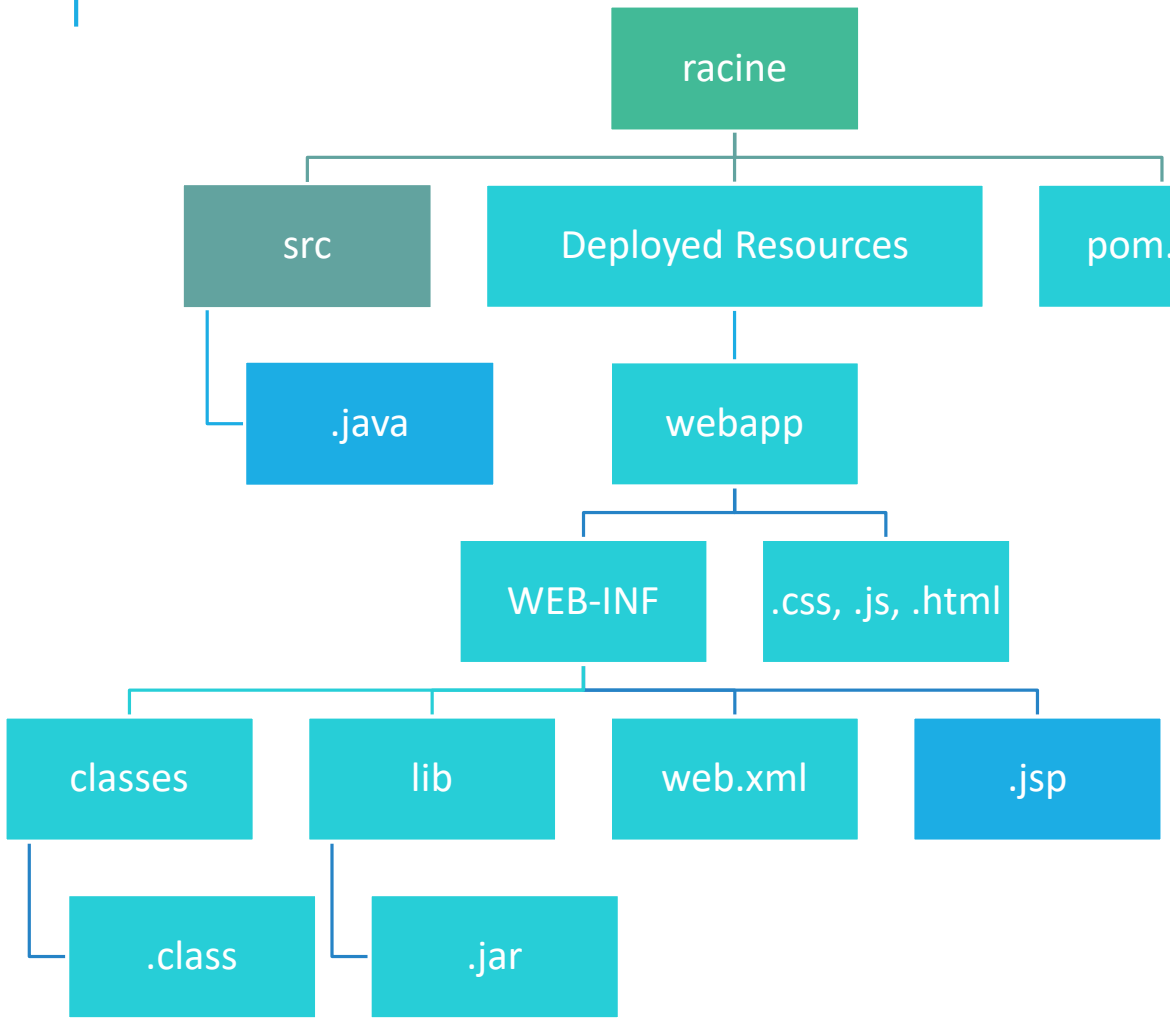


ARCHITECTURE WEB (DÉVELOPPEMENT ECLIPSE)



Le répertoire WebContent est en fait la "racine compilée"

ARCHITECTURE WEB (ECLIPSE / MAVEN WAR)



Le répertoire webapp est en fait la "racine compilée"

LES CONTENEURS

RAPPEL : un conteneur gère le cycle de vie et la vie des instances

- Le conteneur de Servlets gère le cycle de vie des Servlets et tout ce qui s'articule autour

LES CONTENEURS – CONTENEUR DE SERVLET

Initialisation du serveur

- Création du pool de threads auxquels les requêtes seront affectées
- Création des *Servlets* indiquées comme devant être initialisées au démarrage

Première requête

- Chargement de la servlet (si pas déjà fait au démarrage)
- Méthode « init() » invoquée par le conteneur
- Création des objets **Request** et **Response** spécifiques à la requête

Autres requêtes

- La méthode « service() » est invoquée dans une nouvelle thread
- Le conteneur donne les paramètres **Request** et **Response** à la servlet

Déchargement de la servlet

- La méthode « destroy() » est appelée

LES SERVLETS

Classe Java qui traite les requêtes HTTP

- Hérite de la classe abstraite **javax.servlet.http.HttpServlet**

Implémentation (au besoin) des méthodes choisie

- *doGet* Requête HTTP GET
- *doPost* Requête HTTP POST
- *doPut* Requête HTTP PUT
- *doDelete* Requête HTTP DELETE

Chaque *Servlet* doit être mappée sur une URL

- *Sans prendre en compte le nom du serveur ni le nom du projet*

LES SERVLETS

Déclaration des mapping (URL, chemin d'accès web vers la *Servlet*)

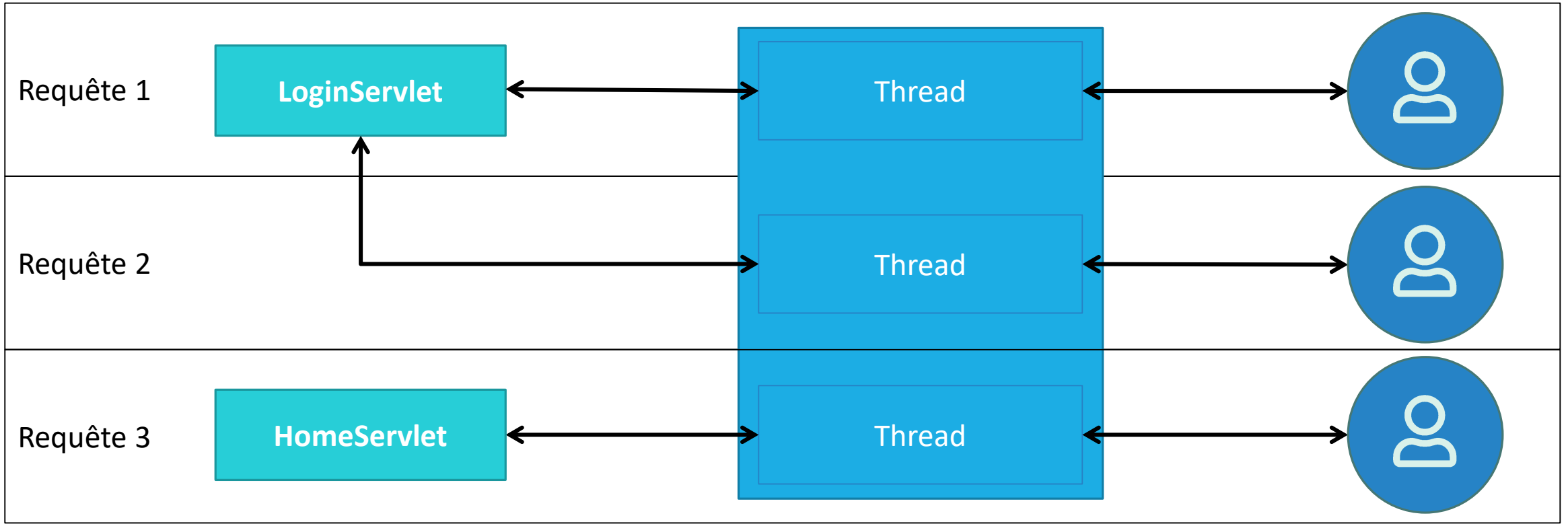
- Configuration « web.xml »

```
<servlet>
  <servlet-name>Home</servlet-name>
  <servlet-class>fr.formation.servlet.HomeServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>Home</servlet-name>
  <url-pattern>/home</url-pattern>
</servlet-mapping>
```

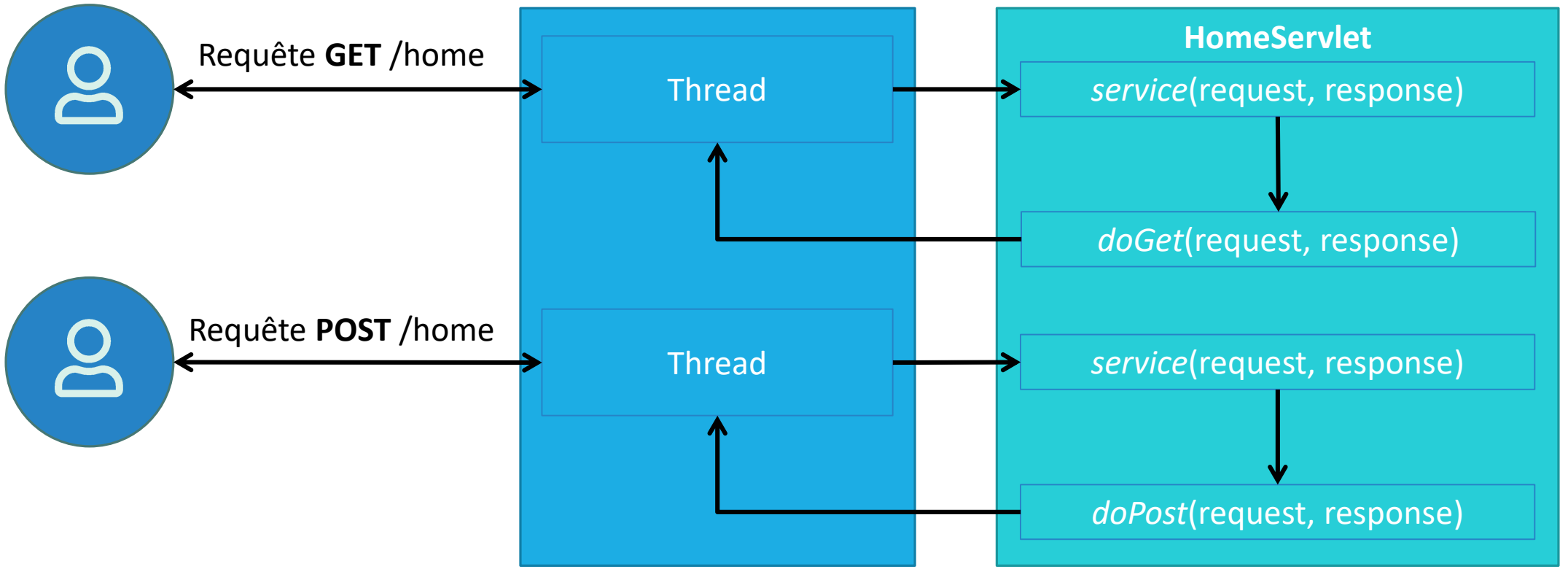
- Annotation

```
@WebServlet("/home")
public class HomeServlet extends HttpServlet { }
```

LES SERVLETS



LES SERVLETS



La *Servlet* **HomeServlet** est mappée sur "/home"

LES SERVLETS

HttpServletRequest et **HttpServletResponse** sont injectés dans les méthodes

- doGet, doPost, …

HttpServletRequest contient des informations sur la requête HTTP

- Les paramètres de la requête
- Les attributs
- La session utilisateur
- *Toute autre information envoyée du client vers le serveur*

HttpServletResponse contient les éléments de la réponse HTTP qui sera envoyée

- C'est avec cet objet qu'on pourra modifier la réponse HTTP
  - Ecrire du contenu HTML par exemple, ou demander une redirection
- *Tout autre élément envoyé du serveur vers le client*

LES SERVLETS

HttpServletRequest

- Récupérer un paramètre de requête
  - http://localhost:8080/mon-projet/home?username=babar

```
String myUsername = req.getParameter("username");
```

- Ajouter un attribut de requête

```
req.setAttribute("nomVariable", "valeur");
```

- Récupérer la session de l'utilisateur

```
Object myVariable = req.getSession();
```

- Récupérer un attribut de la session de l'utilisateur

```
Object myVariable = req.getSession().getAttribute("variableDeSession");
```

LES SERVLETS

HttpServletResponse

- Spécifier le type de contenu (contenu HTML dans l'exemple)

```
resp.setContentType("text/html");
```

- Ajouter du contenu

```
resp.getWriter().println("Bonjour le monde !");
```

- Rediriger vers une autre Servlet

```
resp.sendRedirect("autreServlet");
```

LES SERVLETS – EXERCICE

Créer un nouveau projet « Dynamic Web Project »

Créer une Servlet **HomeServlet**

- La mapper sur « /home »
- La méthode HTTP GET doit retourner un flux HTML « Allô le monde ?! »

Exécuter ce projet sur le serveur Apache Tomcat

LES SERVLETS

On distingue les paramètres des attributs

- Les paramètres sont des chaînes de caractères envoyées par l'utilisateur (toujours dans une requête)
  - GET ou POST
- On ne peut que lire l'information
  - getParameter() sur l'objet **Request**
- Les attributs sont des objets stockés sur le serveur d'application
  - Stockés dans un scope spécifié
- On peut lire et écrire de l'information
  - setAttribute() pour sauvegarder un attribut
  - getAttribute() pour récupérer un attribut

LES SERVLETS – EXERCICE

Modifier la Servlet **HomeServlet**

- Dans la méthode GET, attendre un paramètre « username »
- Retourner un flux HTML « Bonjour "le nom d’utilisateur du paramètre" »

LES SERVLETS

Chaque Servlet peut initialiser des paramètres, en XML

```
<servlet>
  <servlet-name>HomeServlet</servlet-name>
  <servlet-class>fr.formation.servlet.HomeServlet</servlet-class>

  <init-param>
    <param-name>nomParametre</param-name>
    <param-value>valeur du parametre</param-value>
  </init-param>

  <init-param>
    <param-name>nomParametre2</param-name>
    <param-value>valeur paramètre 2</param-value>
  </init-param>
</servlet>
```

LES SERVLETS

Chaque Servlet peut initialiser des paramètres, par annotation

```
@WebServlet(
  urlPatterns = "/home",
  initParams = {
    @WebInitParam(name="param", value="Valeur"),
    @WebInitParam(name="param2", value="Valeur 2")
  }
)
```

LES SERVLETS

Pour récupérer les paramètres, dans une méthode de la *Servlet*

```
ServletConfig myServletConfig = this.getServletConfig();

resp.getWriter().println("<p>" + myServletConfig.getInitParameter("param") + "</p>");

for (Enumeration e = myServletConfig.getInitParameterNames(); e.hasMoreElements();) {
  String myParameterName = (String)e.nextElement();
  resp.getWriter().println("<p>" + myParameterName + " = " + myServletConfig.getInitParameter(myParameterName));
}
```

LES SERVLETS – EXERCICE

Créer une Servlet **ParamsServlet** avec quelques paramètres

- Parcourir la liste des paramètres et les afficher sur la page web

LES JSP / LES VUES

Servlet

```
resp.setContentType("text/html");

resp.getWriter().println("<!DOCTYPE html>");
resp.getWriter().println("<html>");
resp.getWriter().println("<head>");
resp.getWriter().println("<title>Ma première page</title>");
resp.getWriter().println("</head>");
resp.getWriter().println("<body>");
resp.getWriter().println("<p>Allô le monde ?!<p>");
resp.getWriter().println("</body>");
resp.getWriter().println("</html>");
```

JSP

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ma première page</title>
  </head>

  <body>
    <p>Allô le monde ?!<p>
  </body>
</html>
```

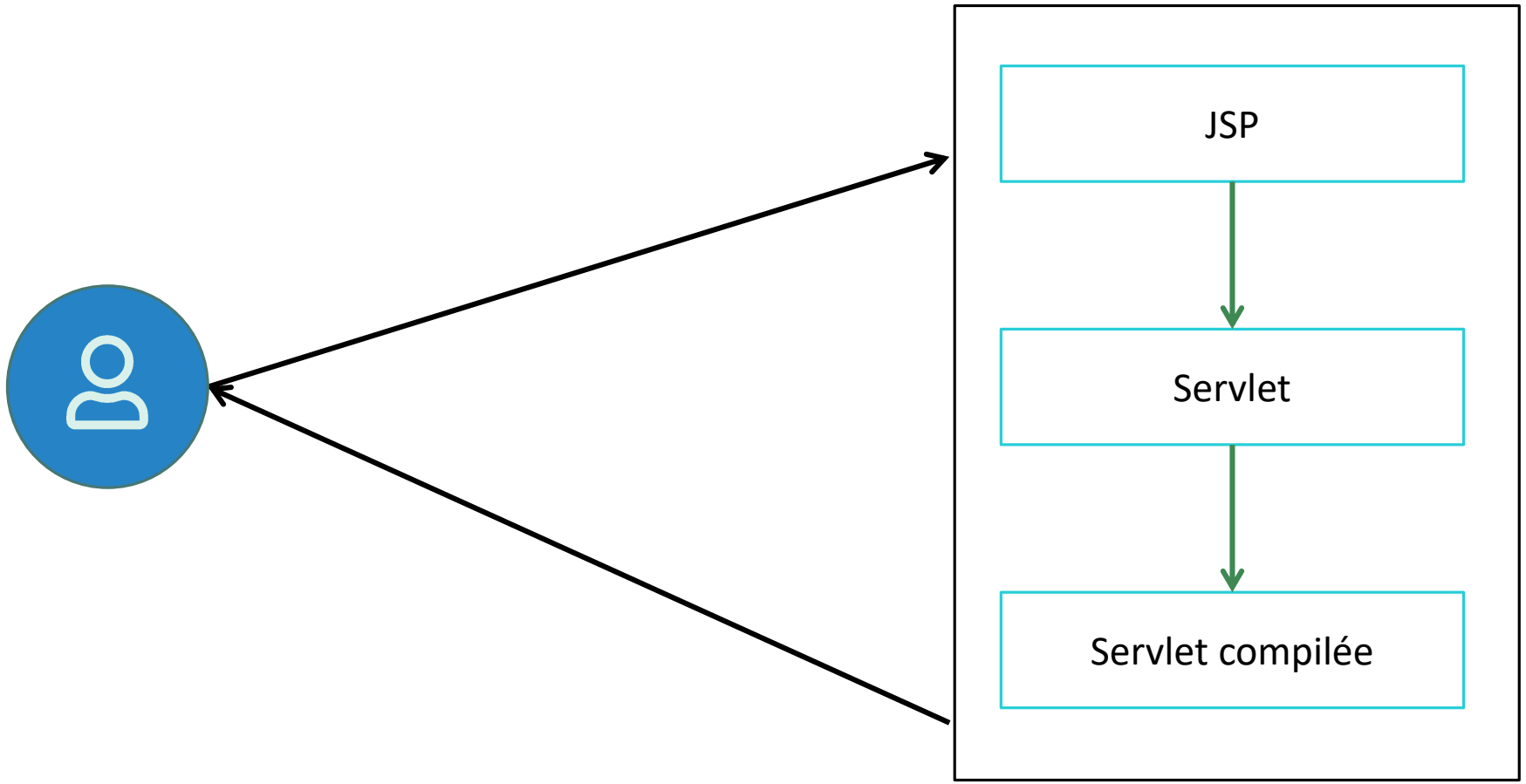
LES JSP / LES VUES

Pénible d'écrire la vue (HTML) dans un fichier Java

JSP est une *Servlet* compilée et auto-indéxée

- Déployée par le conteneur de *Servlet*
- Les variables request et response sont accessibles

LES JSP / LES VUES



LES JSP / LES VUES

Dans une JSP, on peut inclure du Java, ou des instructions pour fabriquer / compiler

- Directive

```
<%@ directive option %>
```

- Code Java (Scriptlet)

```
<% ... %>
```

- Expression (Impression de la valeur d'une variable)

```
<%= ... %>
```

- Commentaire Java

```
<%-- ... --%>
```

- Déclaration

```
<%! ... %>
```

LES JSP / LES VUES

3 directives possibles

- page informations relatives à la page
- include identifie des fichiers à inclure
- taglib indique que la page utilise une bibliothèque de balises (similaire au XMLNS)

Quelques exemples

- Importer une classe

```
<%@ page import="package.class" %>
```

- Préciser l'encodage

```
<%@ page pageEncoding="UTF-8" %>
```

- Ajouter une taglib

```
<%@ taglib uri="http://lurl-de-la-lib" prefix="leprefix" %>
```

LES JSP / LES VUES

On peut utiliser les Expressions Langages (EL) en JSP en utilisant la syntaxe suivante

- ${ … }

```
${ variable }        ${ 5 + 5 }
```

- Permet de lire une variable qui se trouve, **en tant qu'attribut**, dans un des scopes existants
- Pour lire un paramètre de requête
  - Utiliser l'attribut « param »

```
${ param.nomParametre }
```

LES JSP / LES VUES – LES SCOPES

Il existe 4 *scopes* (portées), selon le cycle de vie

Scope / Portée	Servlet	JSP
Application	getServletContext().getAttribute("attr")	applicationScope["attr"]
Session	req.getSession().getAttribute("attr")	sessionScope["attr"]
Request	req.getAttribute("attr")	requestScope["attr"]
Page / Vue	–	pageScope["attr"]

Implémentés grâces aux attributs

- De contexte applicatif
- De session
- De requête

LES JSP / LES VUES – LES SCOPES

Application

- Démarrage de l'application jusqu'à l'arrêt
- Identique à tous les utilisateurs

Session

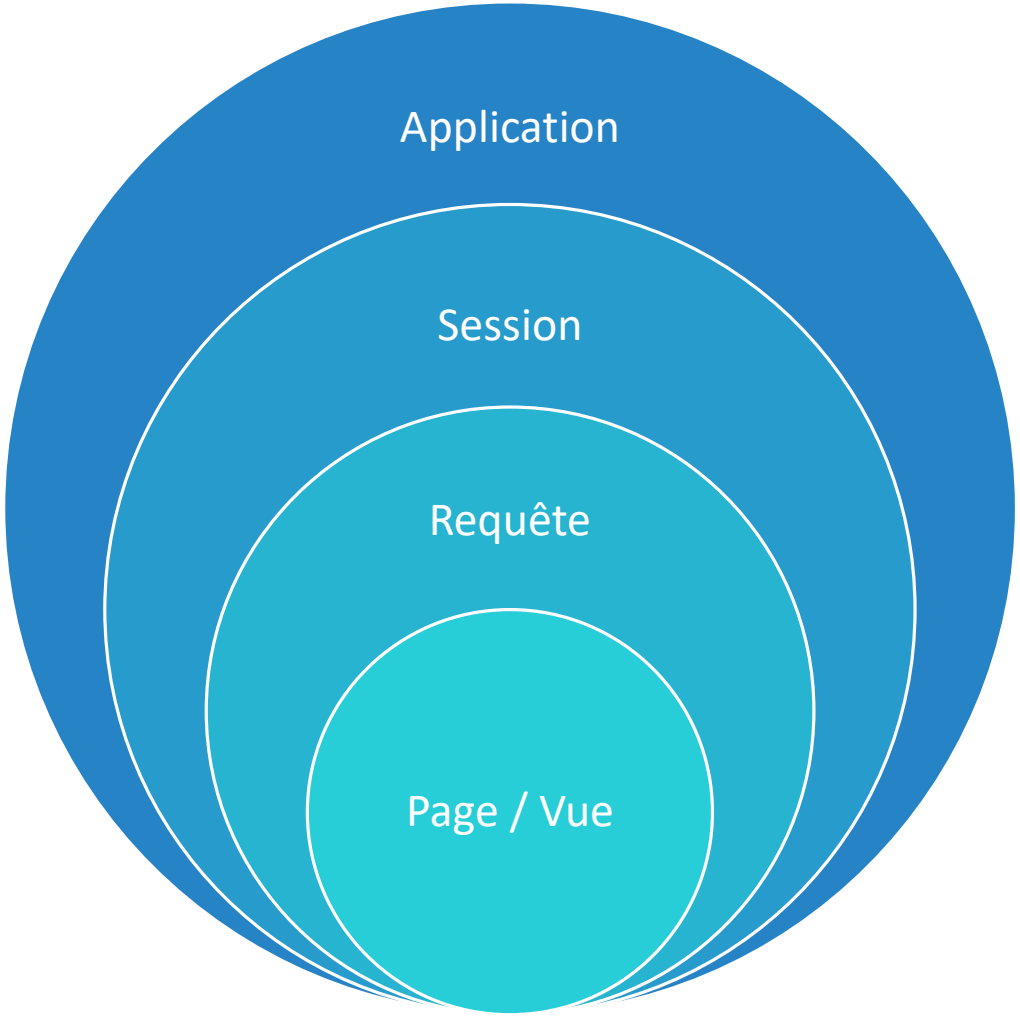
- Démarrage de la session jusqu'à l'inactivité
- Pour chaque utilisateur

Requête

- A chaque requête HTTP

Vue

- A chaque génération de vue



LES JSP / LES VUES – EXERCICE

Créer un fichier JSP « home.jsp » à la racine du projet compilé

- Afficher « Allô le monde ?! »
- Afficher « Bonjour "le nom d’utilisateur du paramètre" »

LES JSP / LES VUES – LA DÉLÉGATION

Pour répondre au modèle MVC

- *Servlet* joue le rôle de Contrôleur
- JSP joue le rôle de la vue

Il faut rendre les pages JSP inaccessibles (les placer dans */WEB-INF/views/*)

Il faut déléguer la requête de la *Servlet* vers la vue JSP

LES JSP / LES VUES – LA DÉLÉGATION

Délégation de transfert

- Contexte de Servlet ➔ Dispatcher ➔ Forward
- Transférer la suite du traitement à une autre *Servlet* ou à une vue JSP

```
this.getServletContext().getRequestDispatcher("/WEB-INF/views/home.jsp").forward(req, resp);
```

Délégation de transmission

- Requête ➔ Ajout d'un attribut (attention à l'ordre, l'attribut doit être inséré avant la délégation !)

```
req.setAttribute("nomUtilisateur", "Jeremy");
```

- Lecture de l'attribut dans la JSP

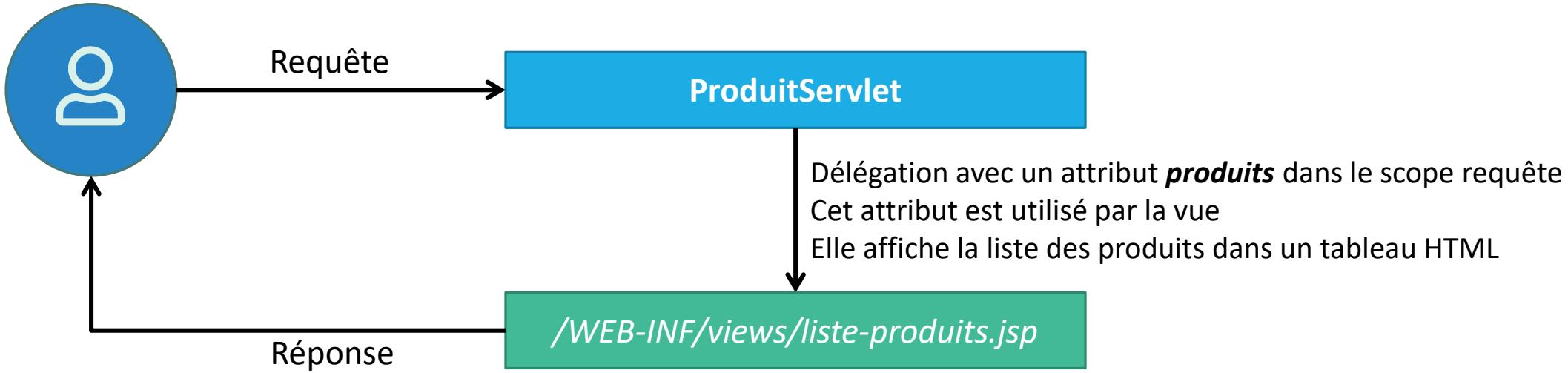
Sans EL

```
<%
String attribut = (String) request.getAttribute("nomUtilisateur");
out.println(attribut);
%>
```

Avec EL

```
${ nomUtilisateur }
```

LES JSP / LES VUES – LA DÉLÉGATION



LES JSP / LES VUES – EXERCICE

Créer un nouveau projet « Maven war » (sans archetype)

- Ajouter la library « Server Runtime » Apache Tomcat au build path du projet
- Ajouter le fichier **web.xml** (Java EE Tools > Generate Deployment Descriptor Stub)

Reprendre la Servlet **HomeServlet**

- Déléguer vers la vue JSP « home.jsp »

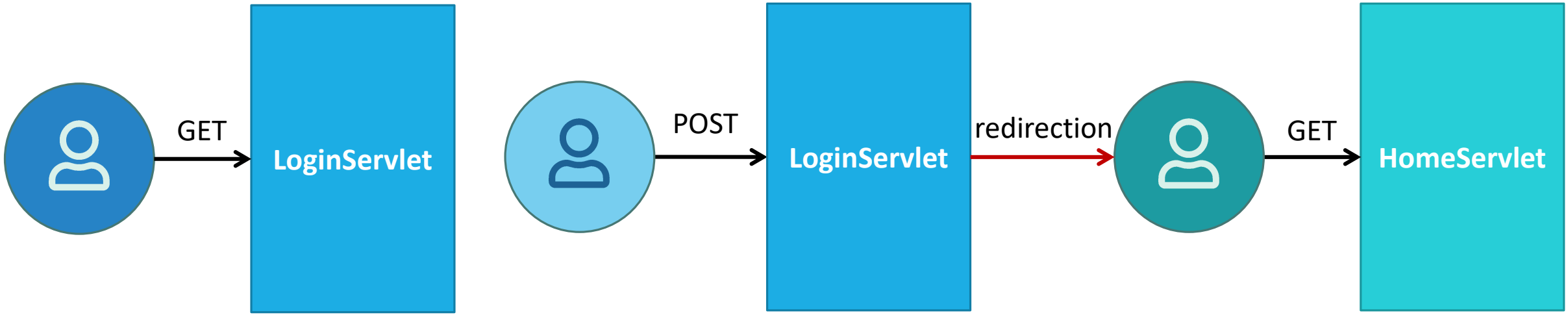
Créer une Servlet **ProduitServlet**

- Utiliser la classe Produit de « eshop-model » (ajouter la dépendance Maven)
- Créer un nouveau Produit à ajouter en tant qu'attribut de la requête
- Afficher le nom du produit dans la JSP

LES JSP / LES VUES – EXERCICE

Créer une Servlet **LoginServlet**

- Afficher un formulaire de connexion (username / password)
- Récupérer le username et le stocker dans une variable de session
- Rediriger vers la Servlet **HomeServlet** après la saisie du formulaire (requête POST)
- La page home.jsp doit afficher le nom d'utilisateur enregistré en session



LES JSP / LES VUES – JSTL

Possible d'étendre le vocabulaire JSP avec JSTL

- JSP Standard Tag Library
- Utilisation d'une taglib
  - Dépendance JSTL de **javax.servlet** (version 1.2)

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

```
<table>
  <c:forEach var="i" begin="0" end="7" step="1">
    <tr>
      <td>${ i }</td>
      <td>${ i * i * i }</td>
    </tr>
  </c:forEach>
</table>
```

LES JSP / LES VUES – JSTL

Servlet

```
List<String> myUtilisateurs = new ArrayList<String>();

myUtilisateurs.add("jeremy");
myUtilisateurs.add("anaïs");
myUtilisateurs.add("jessica");
myUtilisateurs.add("julie");

req.setAttribute("utilisateurs", myUtilisateurs);
```

JSP

```
${ utilisateurs.get(1) }
${ utilisateurs[1] }
${ utilisateurs["1"] }
```

JSP / JSTL

```
<table>
  <c:forEach items="${ utilisateurs }" var="utilisateur">
    <tr>
      <td>${ utilisateur }</td>
    </tr>
  </c:forEach>
</table>
```

LES JSP / LES VUES – EXERCICE

Modifier la Servlet **ProduitServlet**

- Constituer une liste de produits
- Afficher la liste des produits dans un tableau HTML

LES JSP / LES VUES – EXERCICE

Modifier home.jsp (utiliser « c:if »)

- Afficher un message « vous n'êtes pas connecté »
  - Si l'utilisateur n'existe pas en session
- Afficher le reste de la page dans l'autre cas

LES JSP / LES VUES – JSTL – ALLER PLUS LOIN

JSTL comporte 5 bibliothèques de balises

- Core — Actions pour la donnée, les itérations, les conditions, les URL
- Format — Formatage des données (nombres, dates, …)
- XML — Manipulation des données en XML
- SQL — Définition des DataSources et des requêtes SQL
- Function — Actions pour manipuler les chaînes de caractères

LES ECOUTEURS D'ÉVÈNEMENTS

Un *Listener* écoute des évènements de l'API *Servlet* et déclenche des actions

- Evènements Attribut (ajout & retrait)
  - Attribut d'application
  - Attribut de session
  - Attribut de requête
- Evènements Cycle de vie (création & destruction)
  - Contexte d'application
  - Contexte de session
  - Contexte de requête

LES ECOUTEURS D'ÉVÈNEMENTS

Chaque type de *Listener* possède son interface

Type de *Listener*	Interfaces
Attribut Application	*ServletContextAttributeListener*
Attribut Session	*HttpSessionAttributeListener*
Attribut Requête	*ServletRequestAttributeListener*
Contexte Application	*ServletContextListener*
Contexte Session	*HttpSessionListener* *HttpSessionActivationListener*
Contexte Requête	*ServletRequestListener*

Il faut créer une classe qui implémente une ou plusieurs de ces interfaces

LES ECOUTEURS D'ÉVÈNEMENTS

Une fois l'interface implémentée, il faut indiquer au serveur qu'il s'agit d'un *Listener*

- Déclarer la classe dans le *web.xml* (ou via Annotation @**WebListener**)

```
<listener>
  <listener-class>fr.formation.listener.ApplicationDataInitializationListener</listener-class>
</listener>
```

Lorsqu'un évènement a lieu, selon l'interface, la méthode implémentée sera appelée

LES ECOUTEURS D'ÉVÈNEMENTS – EXERCICE

Créer un écouteur qui, au chargement de l'application, crée une liste de produits

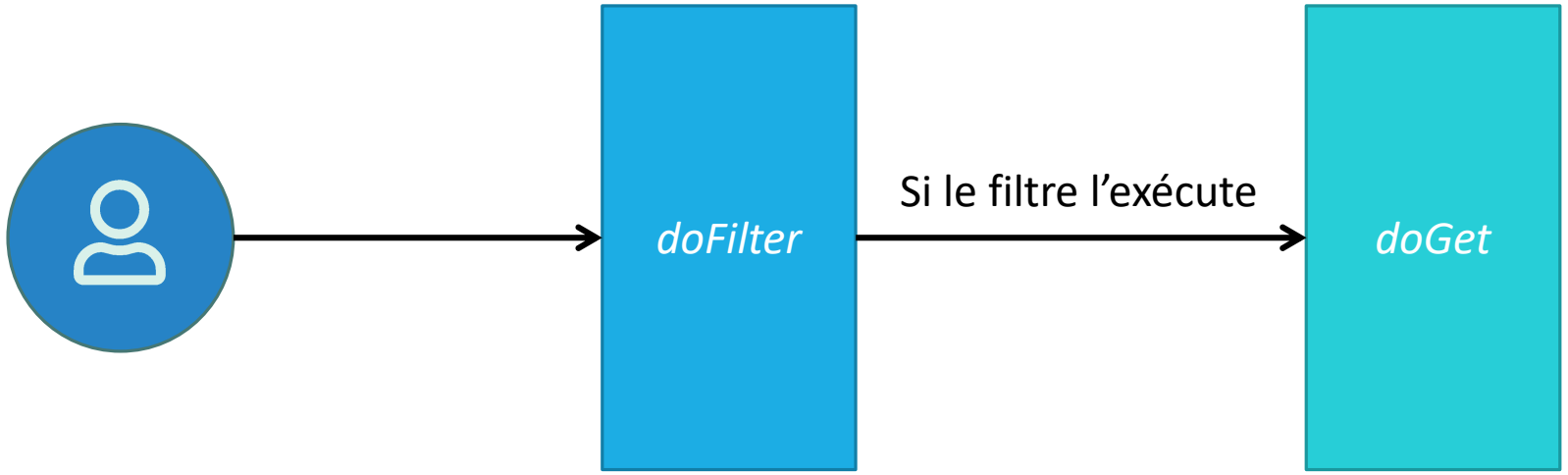
- Utiliser l'interface *ServletContextListener*
- Utiliser ce qui a été fait dans **ProduitServlet**
  - **ProduitServlet** ne doit plus créer la liste des produits
- Afficher la liste des produits dans la page JSP

LES FILTRES

Un *Filter* permet de filtrer une requête HTTP

Il se place avant la *Servlet*

Exemple avec une requête HTTP GET



LES FILTRES

Il faut créer une classe qui implémente l'interface *javax.servlet.Filter*

- La déclarer comme *Filter* dans *web.xml* (ou via Annotation @WebFilter)

Contrairement au *Listener*

- Le *Filter* ne s'applique que sur les requêtes dont le pattern URL correspond à l'URL demandée

```
<filter>
  <display-name>SecuriteFilter</display-name>
  <filter-name>SecuriteFilter</filter-name>
  <filter-class>fr.formation.SecuriteFilter</filter-class>
</filter>

<filter-mapping>
  <filter-name>SecuriteFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

LES FILTRES

La méthode *init* est appelée au démarrage du serveur

La méthode *destroy* est appelée dès que le serveur s'arrête

La méthode *doFilter* est appélée sur chaque pattern d'URL concerné

- *chain.doFilter* permet d'exécuter la *Servlet* concernée

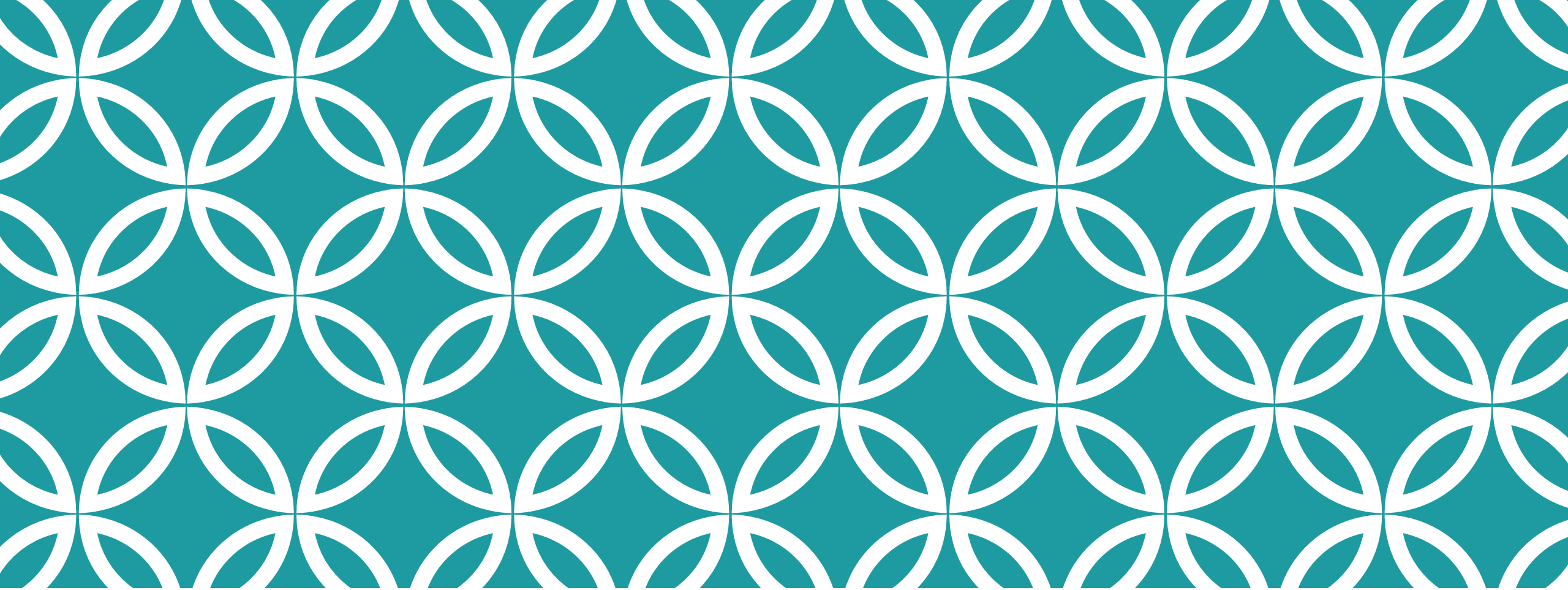
```
public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws IOException, ServletException {
  HttpServletRequest request = (HttpServletRequest)req;
  HttpServletResponse response = (HttpServletResponse)resp;

  /* ... */

  chain.doFilter(request, response);
}
```

LES FILTRES – EXERCICE

Créer un nouveau Filtre **SecuriteFilter**

- Si l'utilisateur n'est pas connecté
  - Refuser l'accès à toutes les pages, sauf la Servlet **LoginServlet**
  - Rediriger vers la page de connexion

- ➔ Utiliser getRequestURI() de l'objet **HttpServletRequests**



EXERCICE SERVLET & JSP

Manipulation Servlet & JSP

EXERCICE

Implémenter Bootstrap

Créer une Servlet et une JSP d'ajout d'un produit

- Le produit s'ajoute à la liste existante

Ajouter dans le tableau HTML de la liste des produits

- Un bouton « éditer »
- Un bouton « supprimer »
- Faire en sorte d'utiliser le même formulaire pour la création et l'édition d'un produit !