

Cours 2

Java EE & Servlets

HEI 2021 / 2022

Bilan des premiers séances

- Vos nouveaux domaines d'expertise :

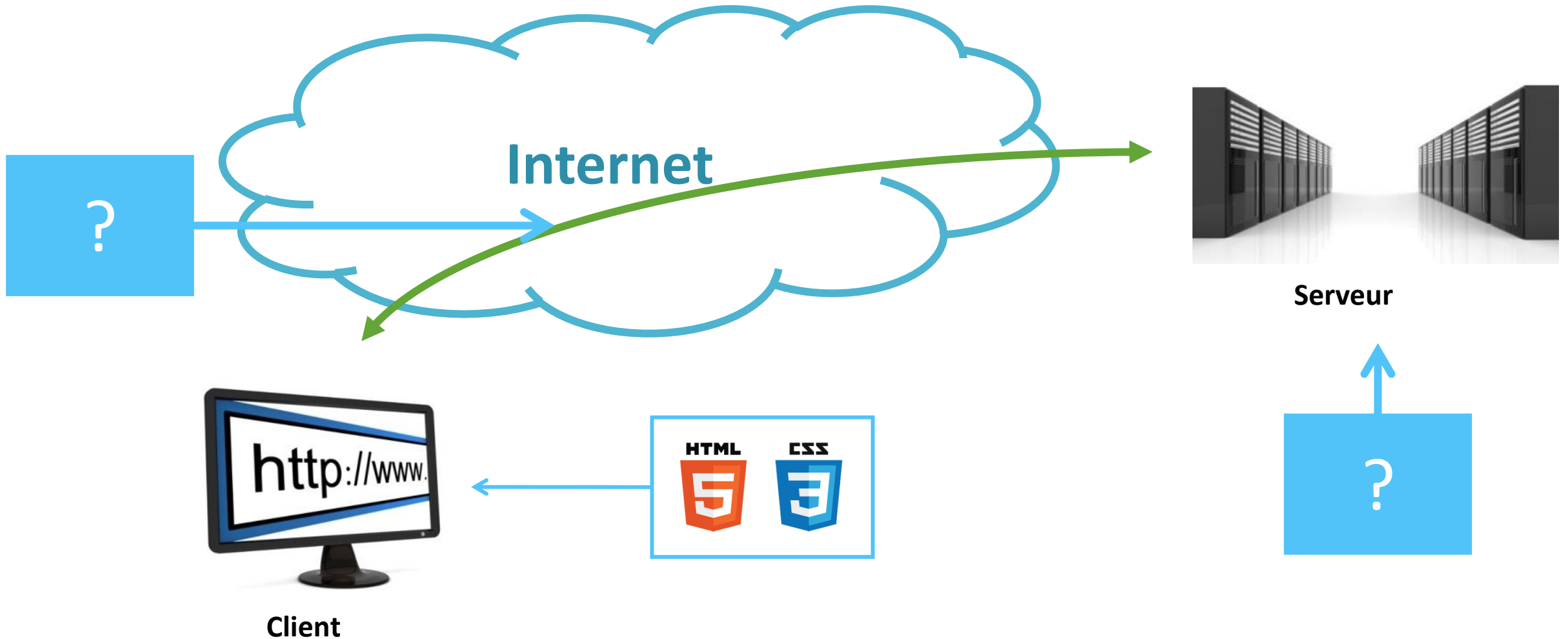
HTML



CSS



Architecture Client-Serveur

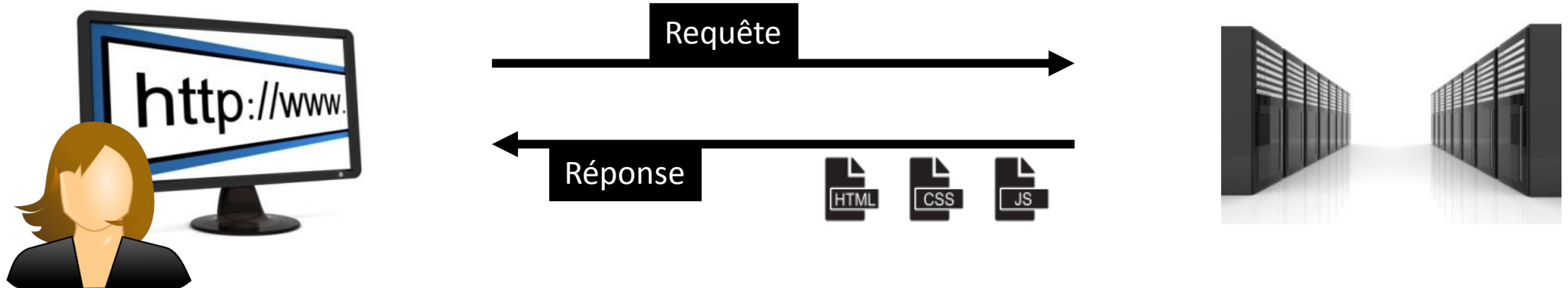


HTTP

Communiquer avec le serveur

Communiquer avec un serveur

- Le principe :



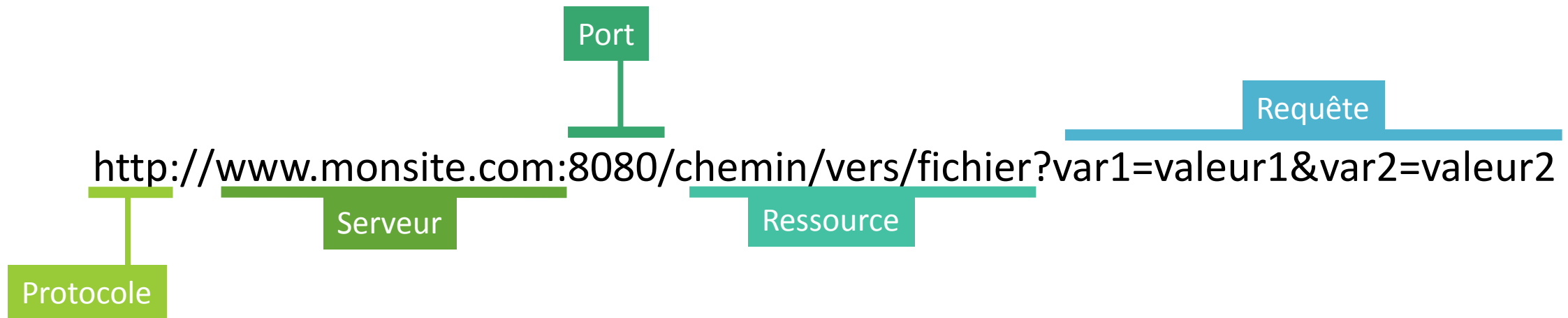
- Le langage d'échange est le protocole HTTP.

Le protocole HTTP

- Hypertext Transfer Protocol
- Protocole réseau sans état (stateless)
 - Chaque requête est indépendante.
- Le port par défaut est 80
- HTTPS est sa version sécurisé
 - Port par défaut : 443

URLs

- HTTP utilise les URL pour identifier la cible de la requête :



Verbes HTTP

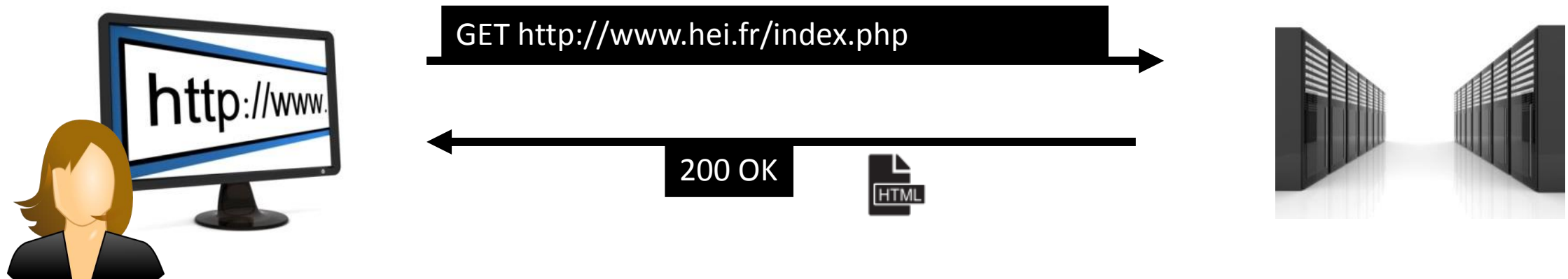
- Une requête HTTP est composée d'une URL et d'un verbe HTTP.
 - URL = la cible = Qui ?
 - Verbe = l'action = Quoi ?
- 4 principaux verbes :
 - Lecture : **GET**
 - Ecriture : **POST**, *PUT*, *DELETE*

Réponse du serveur

- Une réponse HTTP est composée :
 - D'un code de statut (Status code)
 - Du contenu de la réponse (Payload ou Body)
- Le code de statut indique le type de réponse :
 - 2XX = succès
 - 3XX = redirection
 - 4XX = erreur du client
 - 5XX = erreur du serveur

Conclusion

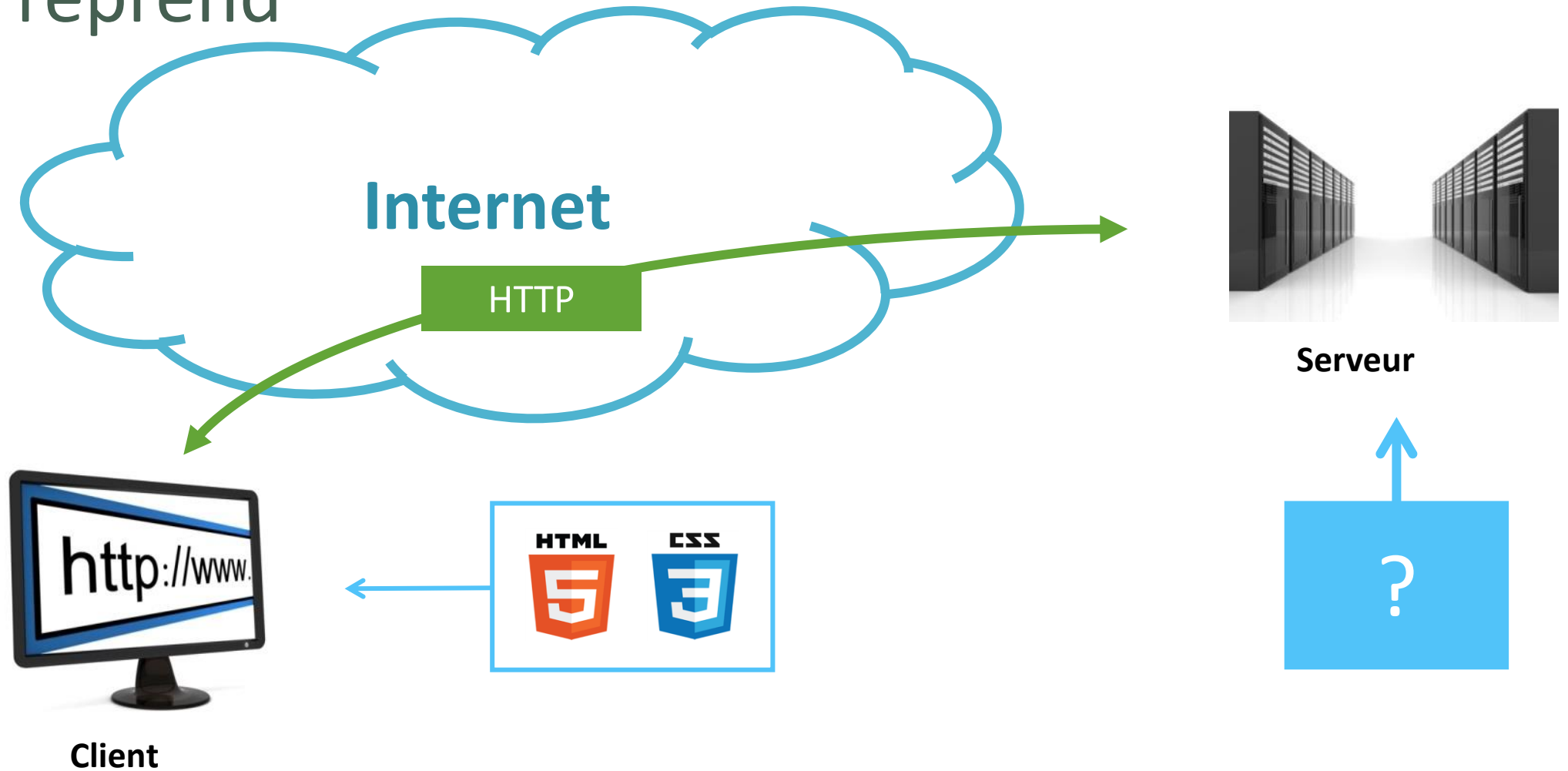
- Si on reprend le schéma du début :



Java EE

Développer en Java côté serveur

Si on reprend

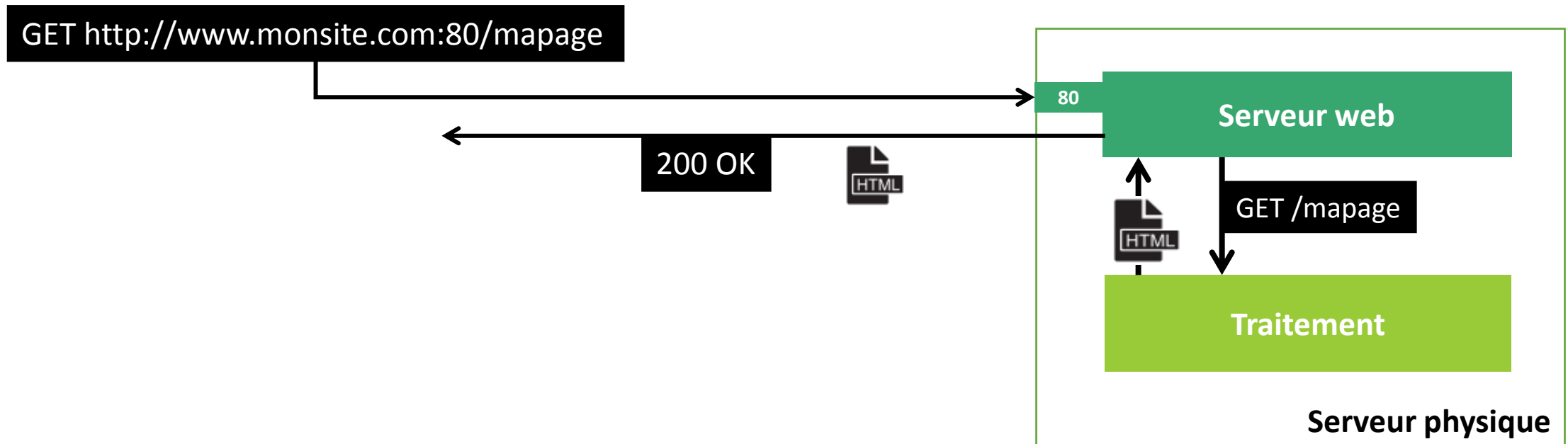


L'intérêt du code côté serveur

- Conserver et manipuler des données
- Partager ces données entre les utilisateurs
- Dynamiser un site web
- Maîtriser l'exécution de son code
- Améliorer la sécurité

Serveur Web

- Un serveur web est une application qui tourne sur le serveur physique distant et qui traite les requêtes HTTP.



Technologies

- Technologies standardisées côté client



Technologies

- Pas de standards, côté serveur



Java EE

- Java Enterprise Edition
 - C'est une extension de Java SE
- On garde le même langage et on y ajoute plein de librairies.

Java EE

JAX-WS

JAX-RS

JSP

JSF

EL

Bean
Validation

Batch
applications

JavaMail

Servlet

JPA

CDI

JAXB

Concurrency
Utilities

Java API for
JSON

JMS

EJB 3

JCA

JTA

JACC

JASPIC

Java API for
WebSocket

Java EE

JAX-WS

JAX-RS

JSP

JSF

EL

Bean
Validation

Batch
applications

JavaMail

Servlet

JPA

CDI

JAXB

Concurrency
Utilities

Java API for
JSON

JMS

EJB 3

JCA

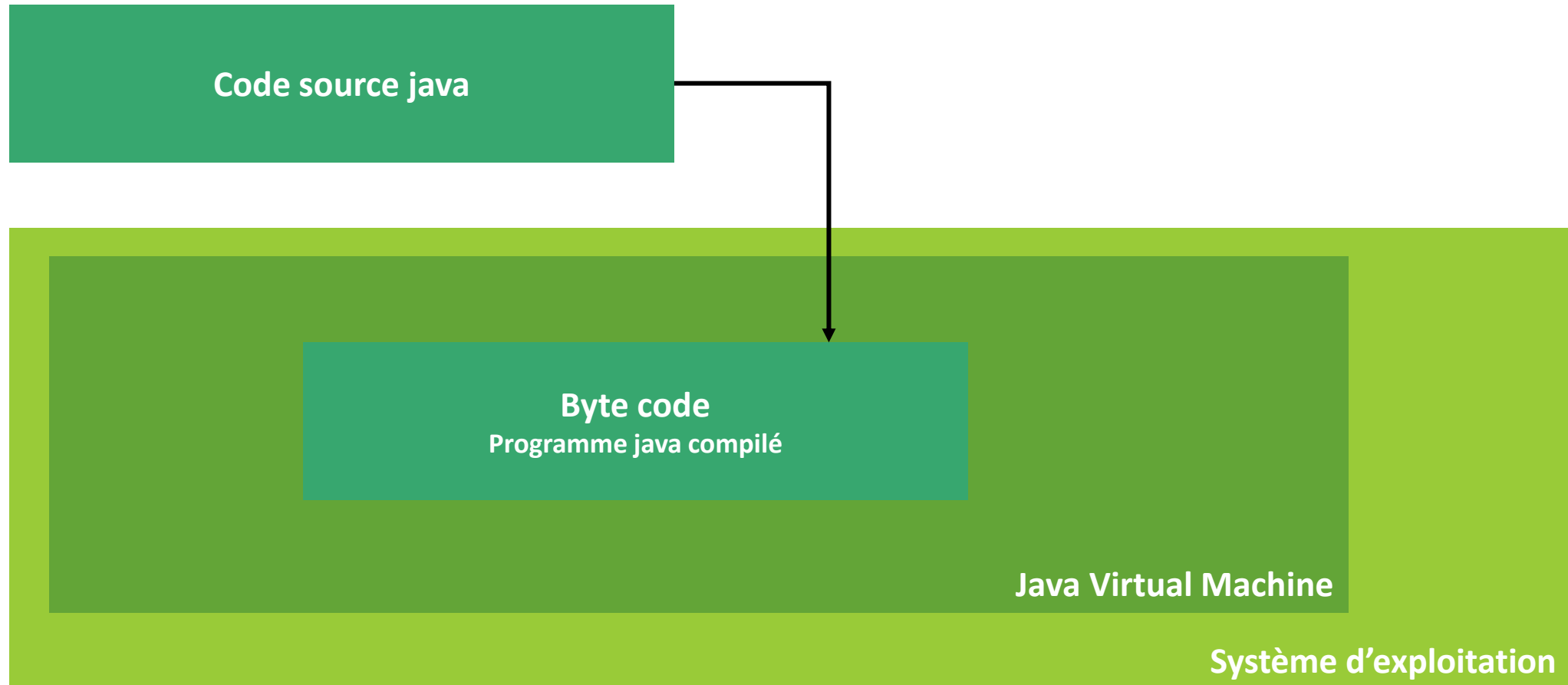
JTA

JACC

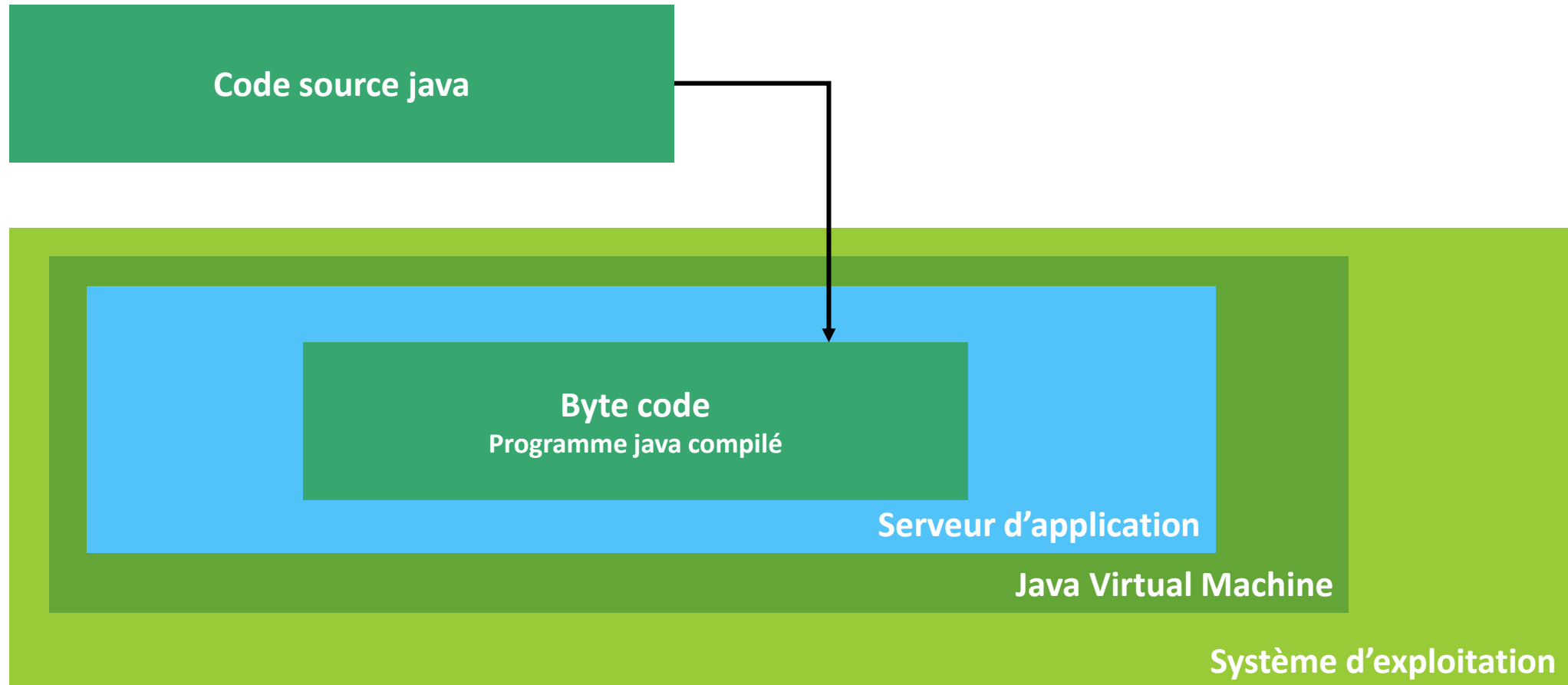
JASPIC

Java API for
WebSocket

Exécution d'un programme Java SE

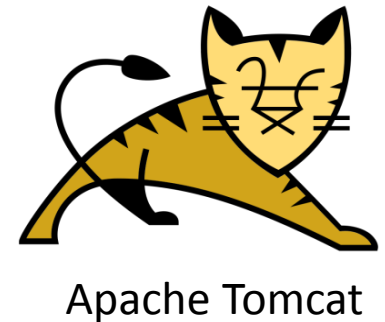


Exécution d'un programme Java EE



Serveur d'application

- Il a la charge d'exécuter l'application
 - Il fournit des librairies Java EE



- Nous allons utiliser Tomcat qui nous servira également de serveur web.

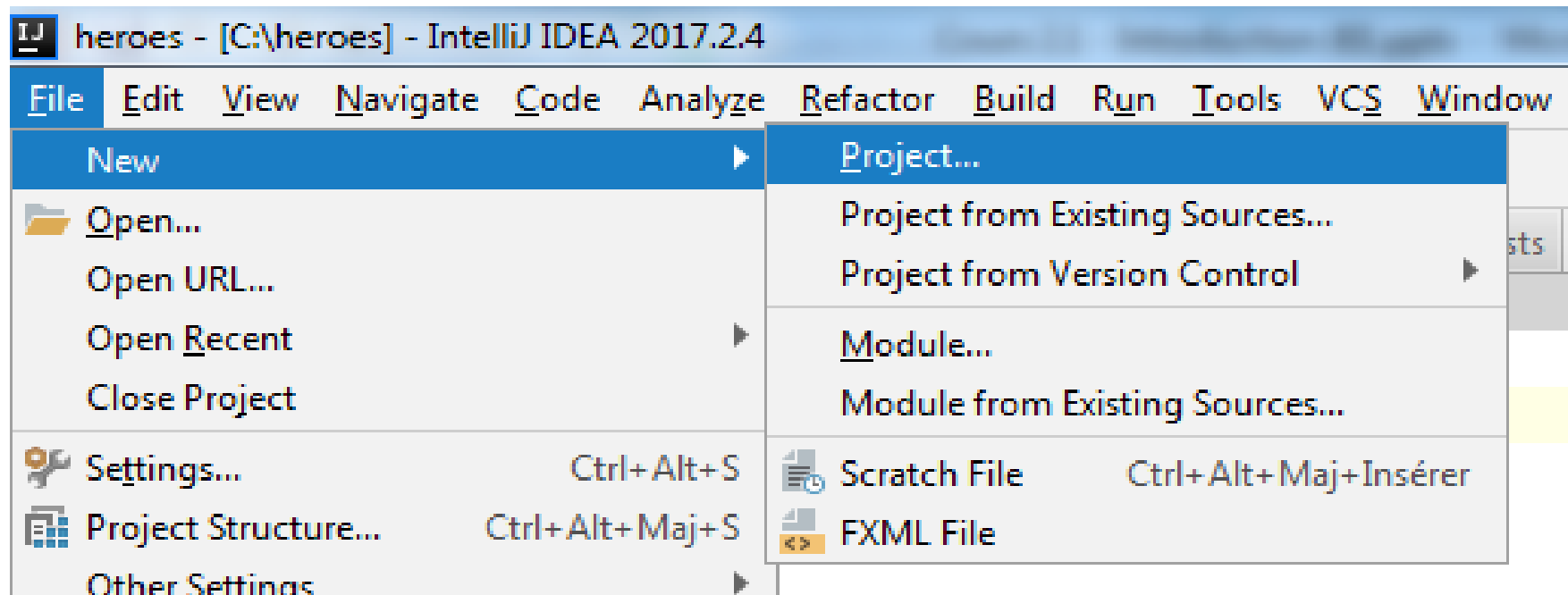
Les outils

- De quoi va-t-on avoir besoin ?
 - De java : **JDK 11**
 - D'un serveur d'application : **Tomcat 9**
 - D'un IDE : **IntelliJ IDEA**
 - D'un navigateur web : **Chrome, Firefox, Edge** ou autre...

Créer sa première webapp

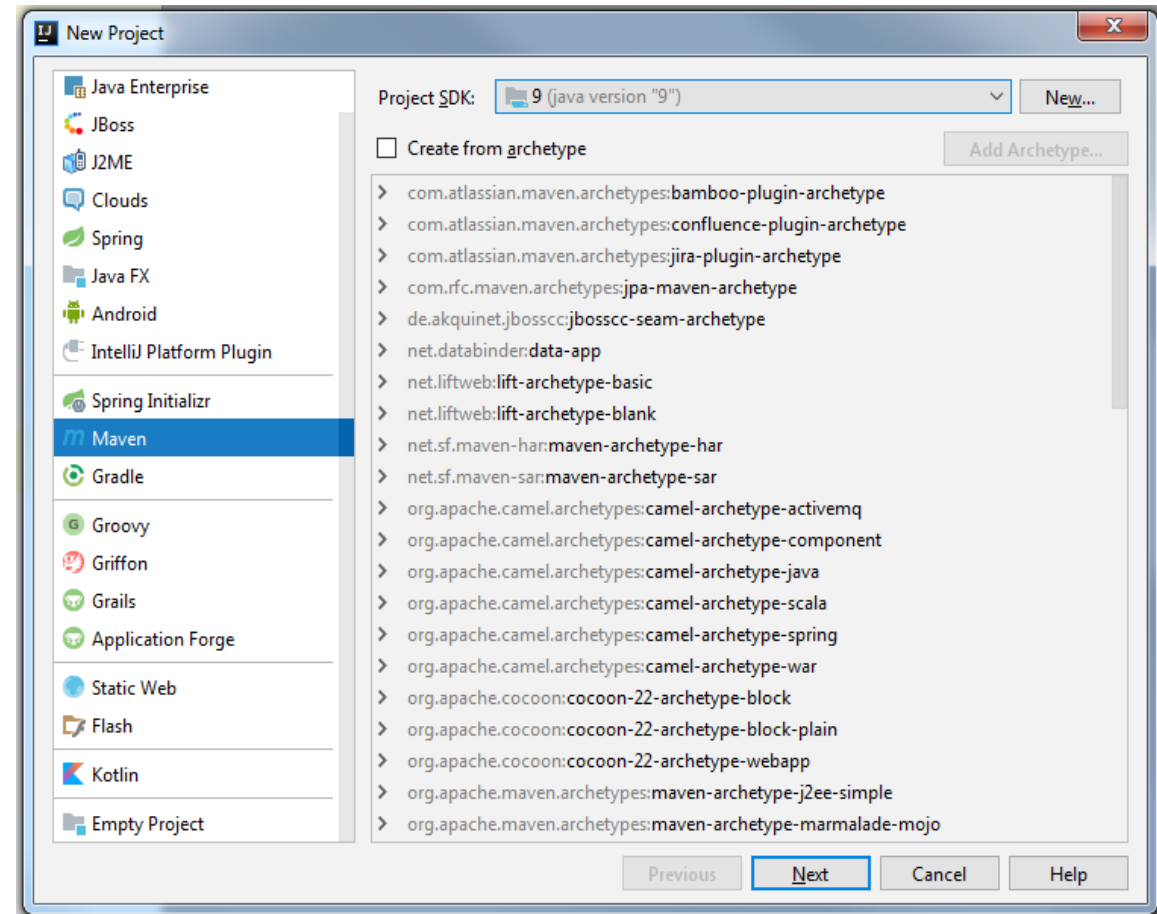
Créer un projet Maven

- Dans IntelliJ, File > new > Project...



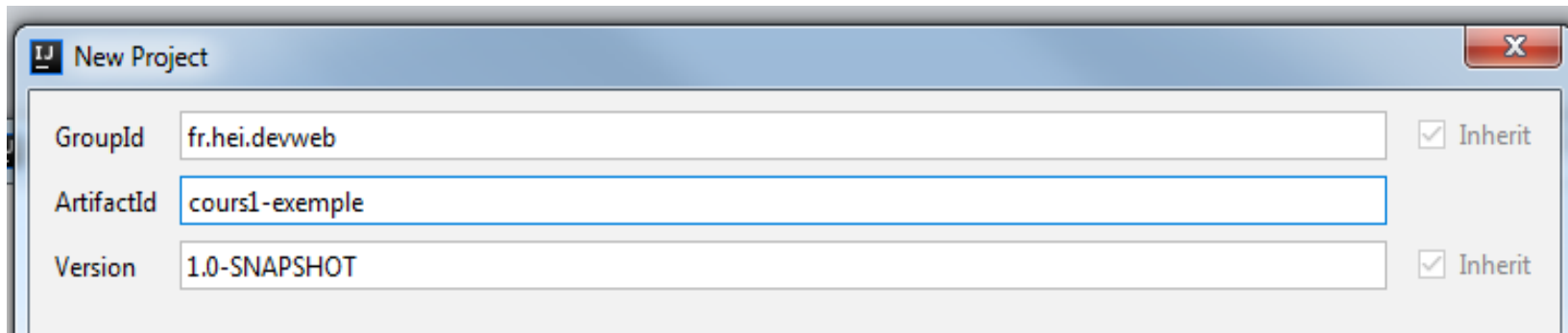
Créer un projet Maven

- Sélectionner « Maven » dans la liste de gauche



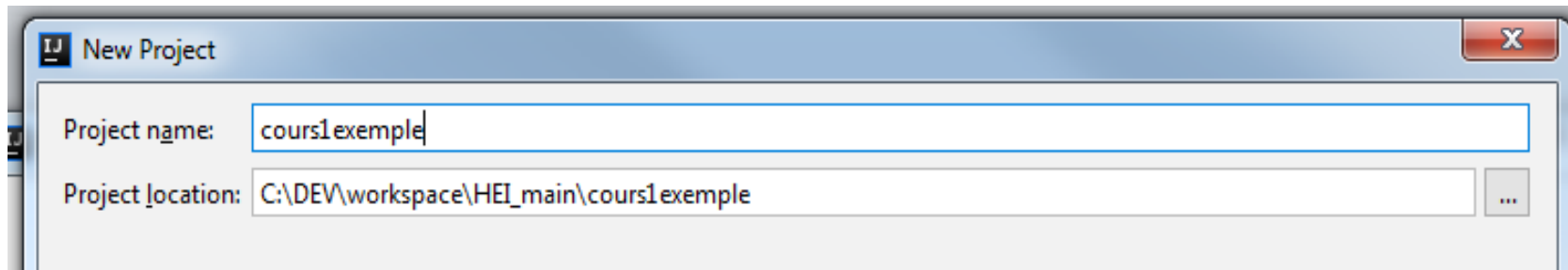
Créer un projet Maven

- Remplir les différents champs :



The screenshot shows the 'New Project' dialog box with the following fields and values:

Field	Value	Checkbox
GroupId	fr.hei.devweb	<input checked="" type="checkbox"/> Inherit
ArtifactId	cours1-exemple	
Version	1.0-SNAPSHOT	<input checked="" type="checkbox"/> Inherit



The screenshot shows the 'New Project' dialog box with the following fields and values:

Field	Value
Project name:	cours1exemple
Project location:	C:\DEV\workspace\HEI_main\cours1exemple

Créer un projet Maven

- Cliquer sur « Finish » et voilà !



Mise à jour du POM.xml

- Par défaut, le projet Maven construit un Jar, il faut lui spécifier qu'on veut générer un War.

```
<groupId>fr.hei.devweb</groupId>  
<artifactId>cours1-exemple</artifactId>  
<version>1.0-SNAPSHOT</version>  
<packaging>war</packaging>
```

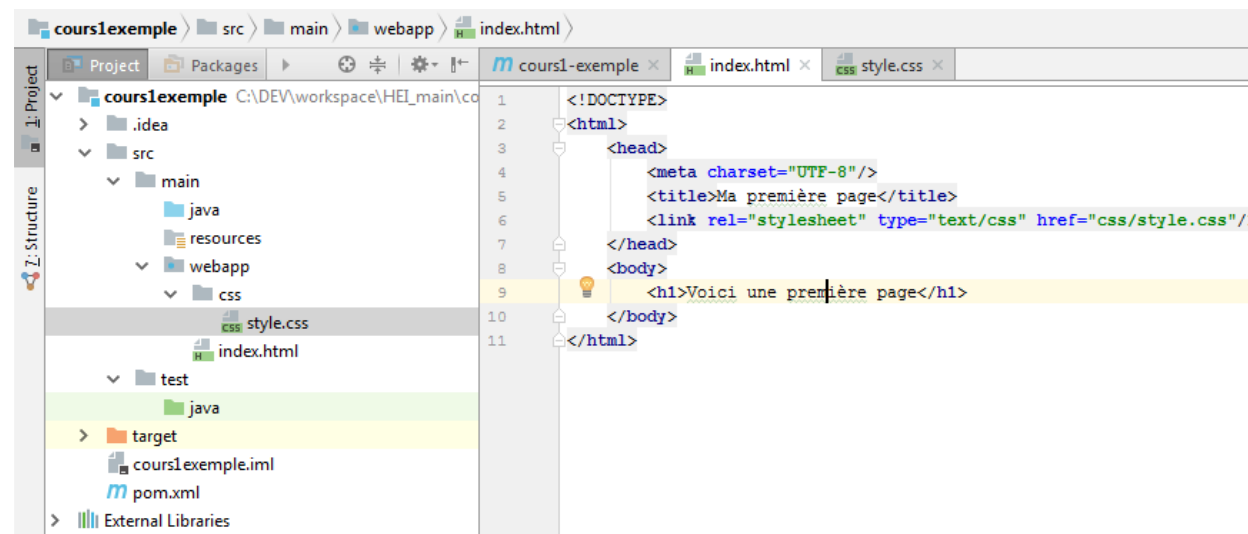
Mise à jour du POM.xml

- Par défaut, Maven compile aussi le projet en Java 5. Pour utiliser Java 11, il est nécessaire de le spécifier.

```
<properties>  
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
  <maven.compiler.source>11</maven.compiler.source>  
  <maven.compiler.target>11</maven.compiler.target>  
</properties>
```

Une webapp en java

- Maven considère que **src/main/webapp** contient notre webapp
- Tout fichier présent dans ce dossier sera accessible dans le navigateur
- Créons quelques fichiers statiques :



Une webapp en java

- Un fichier essentiel d'une webapp en java est le fichier **web.xml**
- Il doit être placé dans le répertoire suivant :
 - /src/main/webapp/WEB-INF/web.xml
- C'est le point d'entrée de notre application pour le serveur d'application.
 - Similaire à la méthode **main()** d'une application java classique.

web.xml

- Le fichier minimal est le suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="4.0" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
        http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd">

</web-app>
```

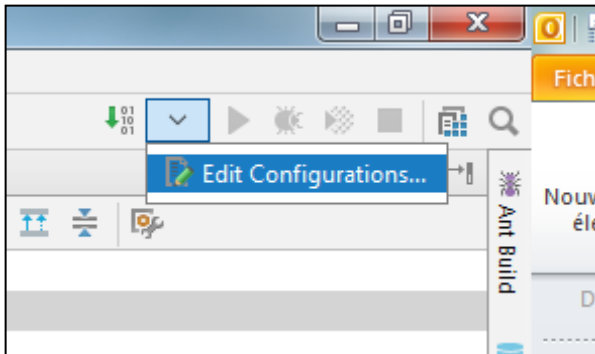
Déploiement

- Pour que notre projet soit accessible sur un navigateur, il faut le déployer au sein d'un serveur d'application.
- Nous allons utiliser Apache Tomcat.

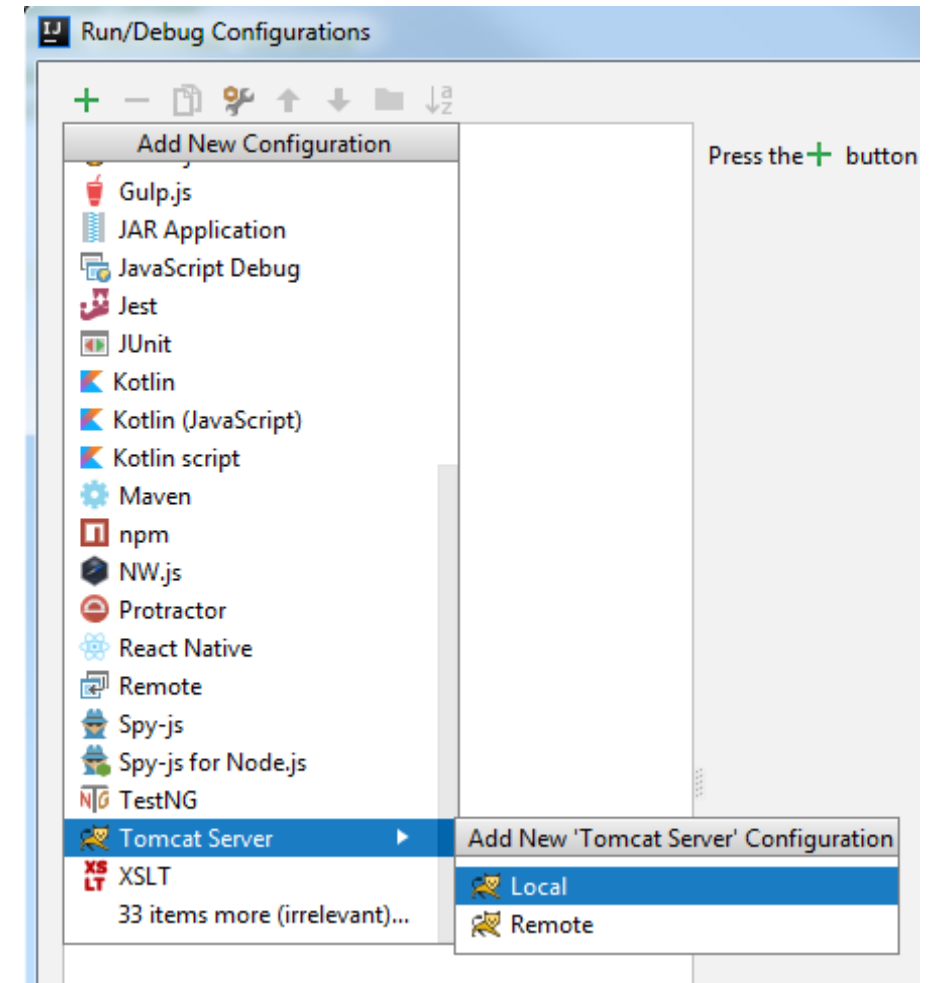


Créer un serveur

- En haut à droite de l'IDE, sélectionner « Edit configurations... »

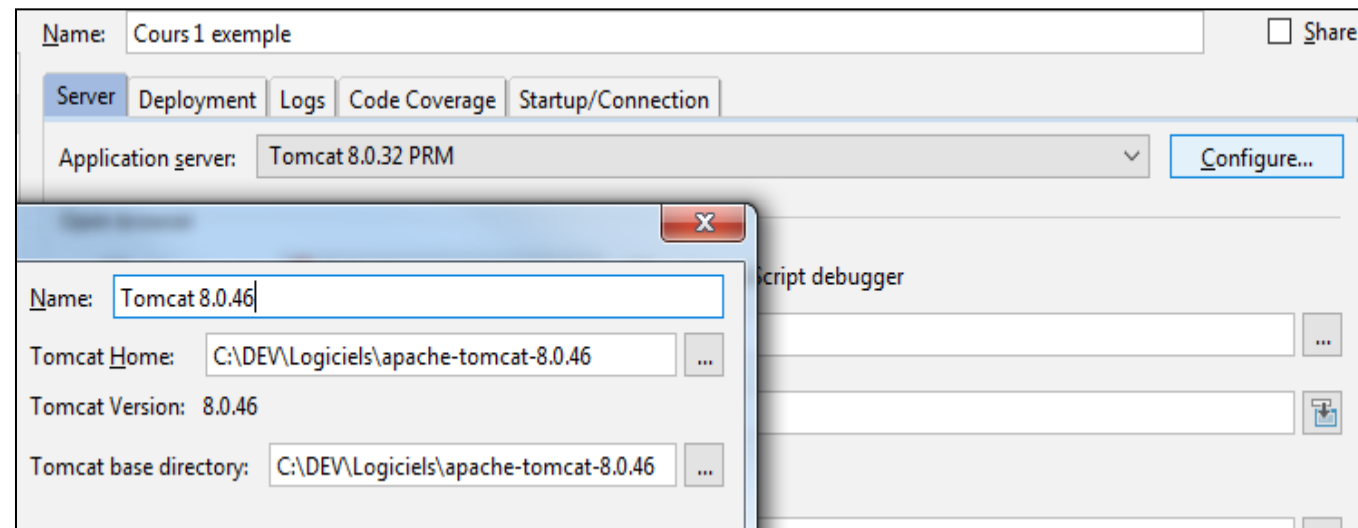


- Puis créer une nouvelle configuration en sélectionnant un serveur Tomcat local.



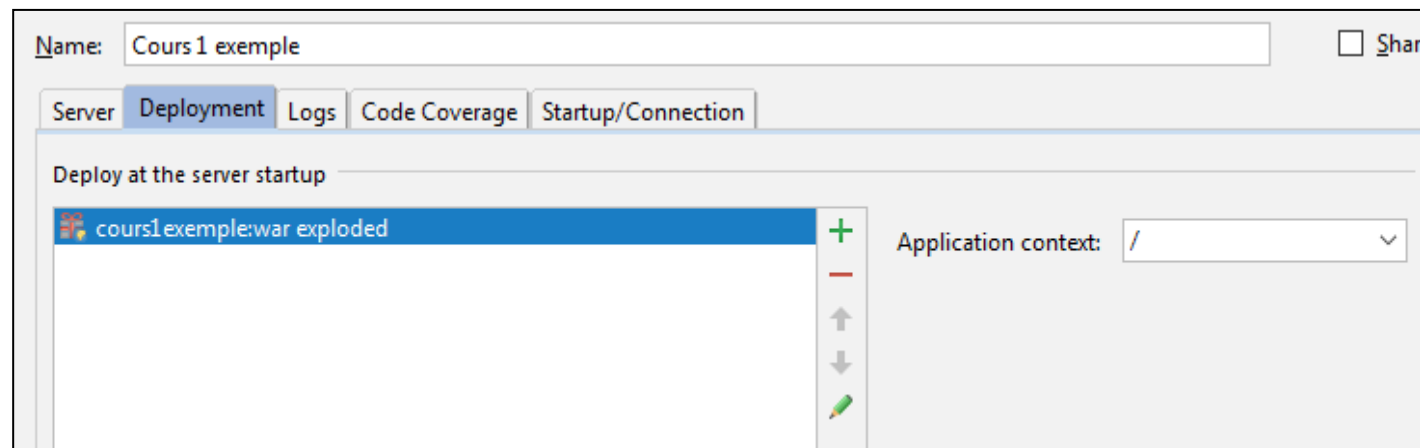
Créer un serveur

- Pour lier IntelliJ avec le Tomcat téléchargé et dézippé sur l'ordinateur, cliquer sur « Configure » puis renseigner les informations nécessaires :



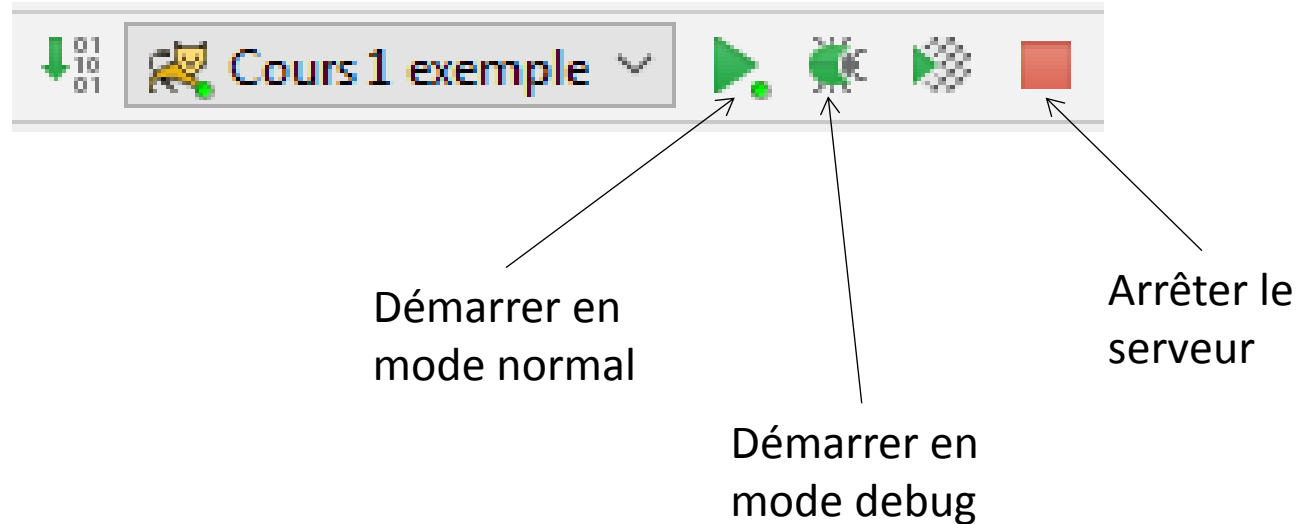
Ajouter notre projet au serveur

- Cliquer sur l'onglet « Deployment » puis sur le « + » pour ajouter votre projet.
- Sélectionner le nom de votre projet suivi de « war exploded ».

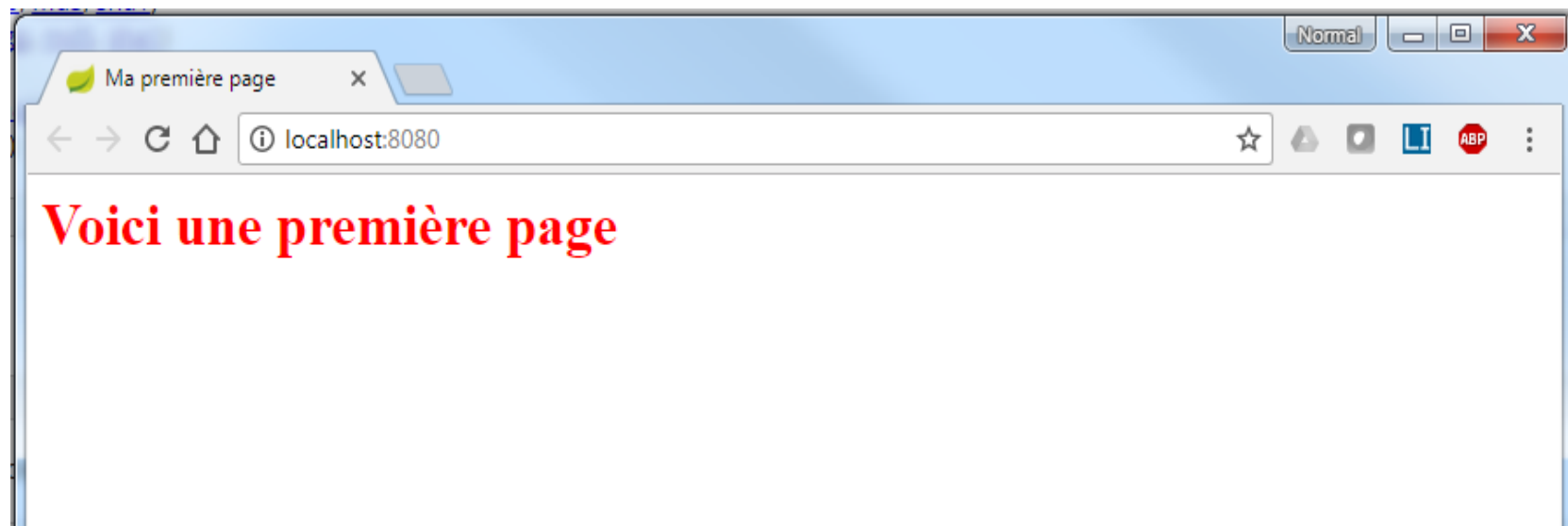


Démarrer le serveur

- Dans le coin en haut à droite de l'IDE, il est maintenant possible de démarrer le serveur.



Ma première webapp :-)

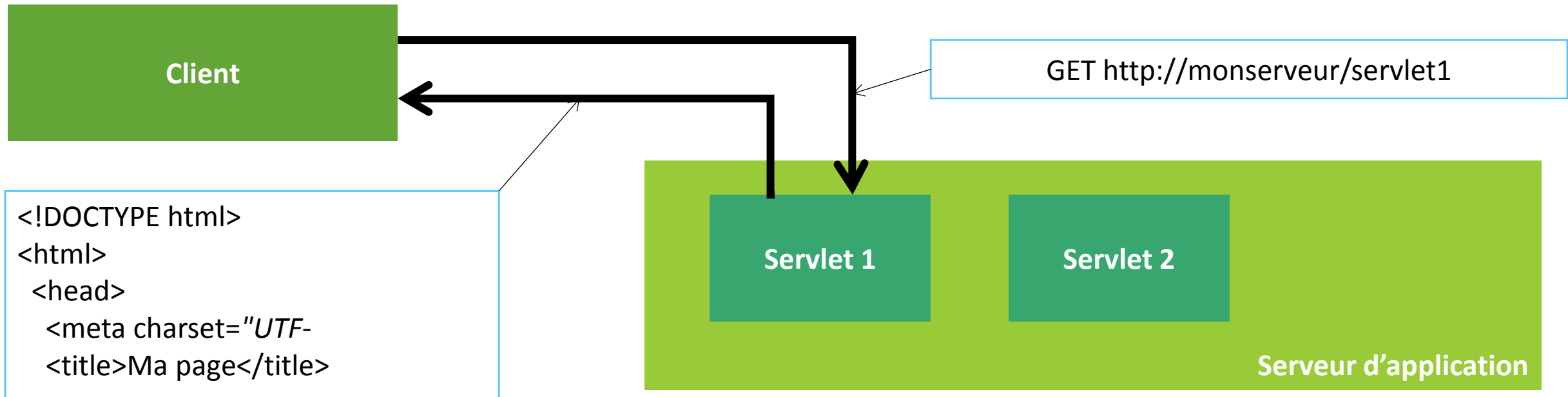


Servlets

Gérer des requêtes HTTP en java

Servlets

- Une servlet est un composant utilisé dans un serveur d'application.
- Elle est invoquée par un navigateur via une requête HTTP.



API Servlets

- L'API Servlets est un ensemble de classes et interfaces Java
 - Packages : `javax.servlet` et `javax.servlet.http`
- Ces classes sont disponibles dans la dépendance maven `servlet-api`.
 - On utilisera la version 4.0.1

Servlet

- Une servlet est une classe java qui hérite de la classe abstraite `HttpServlet`
- 3 méthodes principales :
 - `init()` : appelée à l'instanciation de la servlet par tomcat
 - `service()` : appelée lorsqu'une requête arrive. C'est la méthode qui traite la requête et génère la réponse
 - `destroy()` : appelée à la destruction de la servlet par tomcat

HttpServlet

- Par défaut la méthode `service()` de `HttpServlet` appelle différentes méthodes suivant le type de requête HTTP.
 - HTTP GET : `doGet()`
 - HTTP POST : `doPost()`
 - ...
- Toutes ces méthodes répondent par défaut par une erreur 405 : « method not supported ».

```
public class BonjourServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<meta charset=\"UTF-8\">");
        out.println("<title>Ma première servlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Bienvenue sur ma première servlet !</h1>");
        out.println("</body>");
        out.println("</html>");
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        doGet(request, response);
    }
}
```

Lier notre servlet à une URL

- Actuellement notre servlet est juste une classe présente dans notre application.
 - Elle n'est pas accessible.
- Il faut lier cette servlet à une URL. Il existe 2 façons pour le faire :
 - Avec un fichier de configuration XML
 - Avec des annotations Java

Le fichier web.xml

- La première solution pour déclarer une servlet est d'utiliser le fichier `web.xml`.
- Le fichier `web.xml` doit se situer au chemin suivant :
 - `src/main/webapp/WEB-INF/web.xml`

Le fichier web.xml

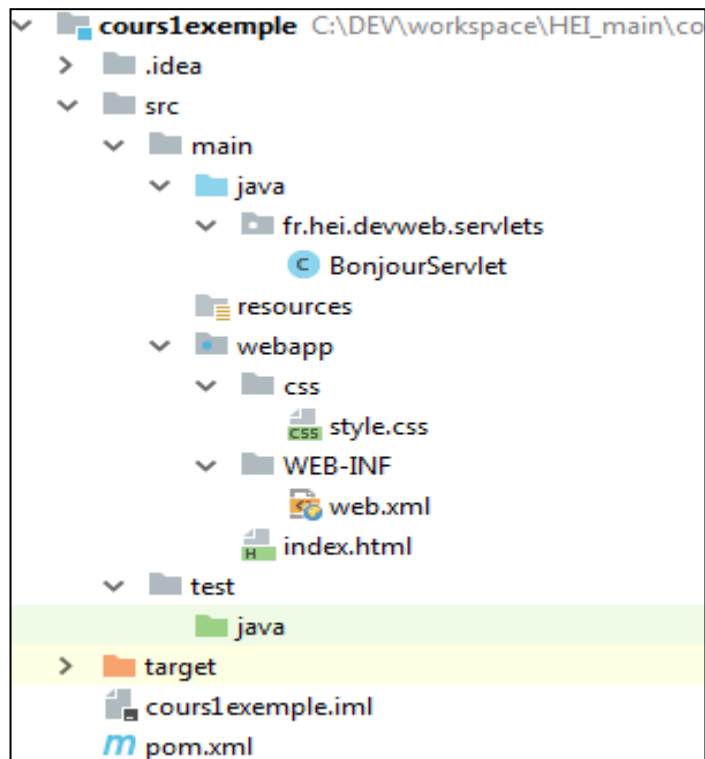
- Fichier XML d'élément racine `<web-app>`
- 2 types d'élément :
 - `<servlet>` : déclare une servlet en lui donnant un nom `<servlet-name>` et en la liant à une classe java `<servlet-class>`
 - `<servlet-mapping>` : associe une URL `<url-pattern>` à une servlet `<servlet-name>`

Le fichier web.xml

- Notre web.xml pour notre première servlet ressemble donc à :

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="4.0" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd">
  <servlet>
    <servlet-name>Bonjour</servlet-name>
    <servlet-class>hei.devweb.servlets.BonjourServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Bonjour</servlet-name>
    <url-pattern>/bonjour</url-pattern>
  </servlet-mapping>
</web-app>
```

Testons cette servlet



- Il suffit de (re)démarrer le serveur Tomcat pour tester la servlet.
- Une connexion à l'URL **/bonjour** dans un navigateur web affiche la page générée par notre servlet.

Plus simple avec les annotations

- Les dernières versions de JavaEE permet de se passer du fichier `web.xml`
- La déclaration des servlets se fait avec l'annotation `@webServlet`

```
@WebServlet("/bonjour")
public class BonjourServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        // ...
    }
}
```

La classe HttpServletRequest

- Permet d'accéder aux paramètres HTTP
 - GET : `http://monserveur/mapage?param=valeur`
 - POST : dans le corps de la requête HTTP
- Ce sont notamment les paramètres envoyés par l'utilisateur par le biais de formulaire HTML.
- Méthodes :
 - `request.getParameter(String)`
 - `request.getParameterMap()`

La classe HttpServletRequest

- Permet également d'accéder :
 - À l'en-tête (header) de la requête HTTP
 - request.getHeader(String)
 - request.getMethod()
 - ...
 - À l'URL
 - À des données de l'utilisateur
 - request.getRemoteAddr()

Une servlet avec un formulaire

```
public class CouleurPrefereeServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<meta charset=\"UTF-8\">");
        out.println("<title>Ma couleur préférée</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Quelle est votre couleur préférée ?</h1>");
        out.println("<form method=\"post\">");
        out.println("<input type=\"text\" name=\"couleur\">");
        out.println("<input type=\"submit\" value=\"Envoyer\">");
        out.println("</form>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

Une servlet avec un formulaire

@Override

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    out.println("<!DOCTYPE html>");
    out.println("<html><head>");
    out.println("<meta charset=\"UTF-8\">");
    out.println("<title>Ma couleur préférée</title>");
    out.println("</head><body>");
    out.println("<h1>Votre couleur préférée :</h1>");
    String couleurSelectionnee = request.getParameter("couleur");
    if(couleurSelectionnee == null || "".equals(couleurSelectionnee)) {
        out.println("<p>Vous n'avez pas sélectionné de couleur. :'(</p>");
    } else {
        out.println(String.format("<p>Votre couleur préférée est : %s</p>",
            couleurSelectionnee));
    }
    out.println("</body>");
    out.println("</html>");
}
```

HttpSession

- Une session permet de suivre un utilisateur pendant sa visite sur notre site.
 - Elle peut être maintenue pendant plusieurs jours.
- C'est le serveur qui gère la session, nous n'avons pas besoin de la gérer nous même (création, suppression...)
- L'objet HttpSession est accessible par la méthode `getSession()` de l'objet HttpServletRequest,

HttpSession

- Les principales méthodes de l'objet sont :
 - `setAttribute(String, Object)` : ajoute un objet en session
 - `getAttribute(String)` : récupère un objet mis en session
- Les objets mis en session doivent être sérialisables (*implements Serializable*)
- Attention à ne pas trop abuser de la session, les données sont conservées en mémoire.

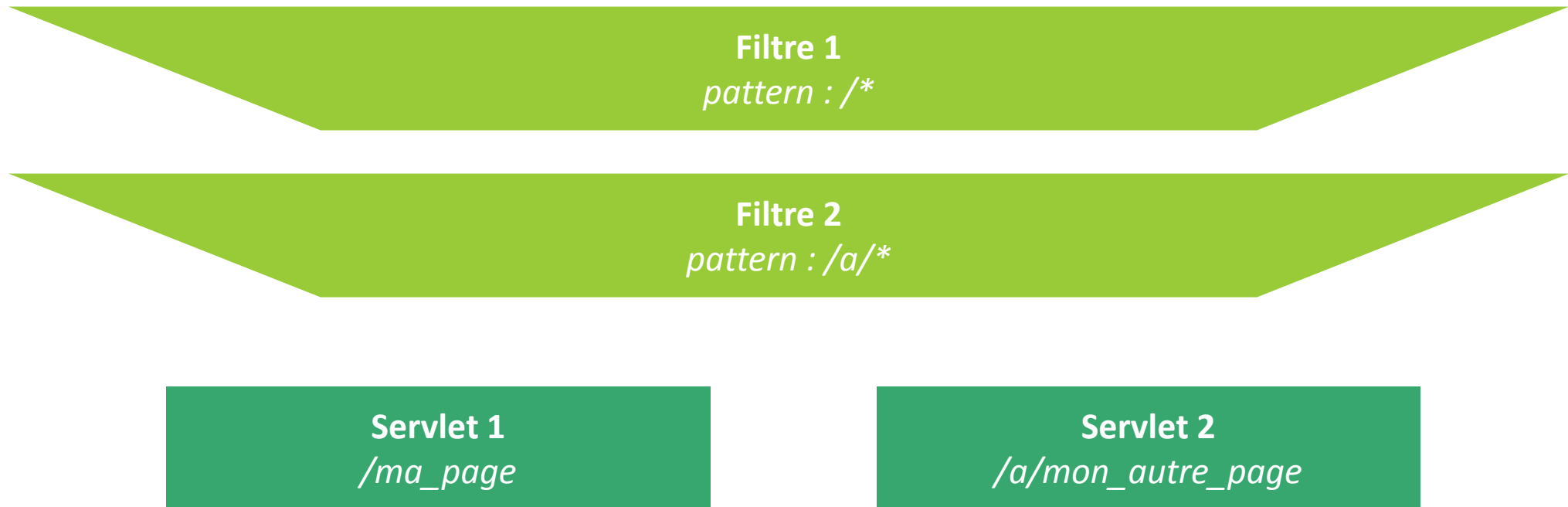
Filtres

Filtrer des requêtes HTTP en java

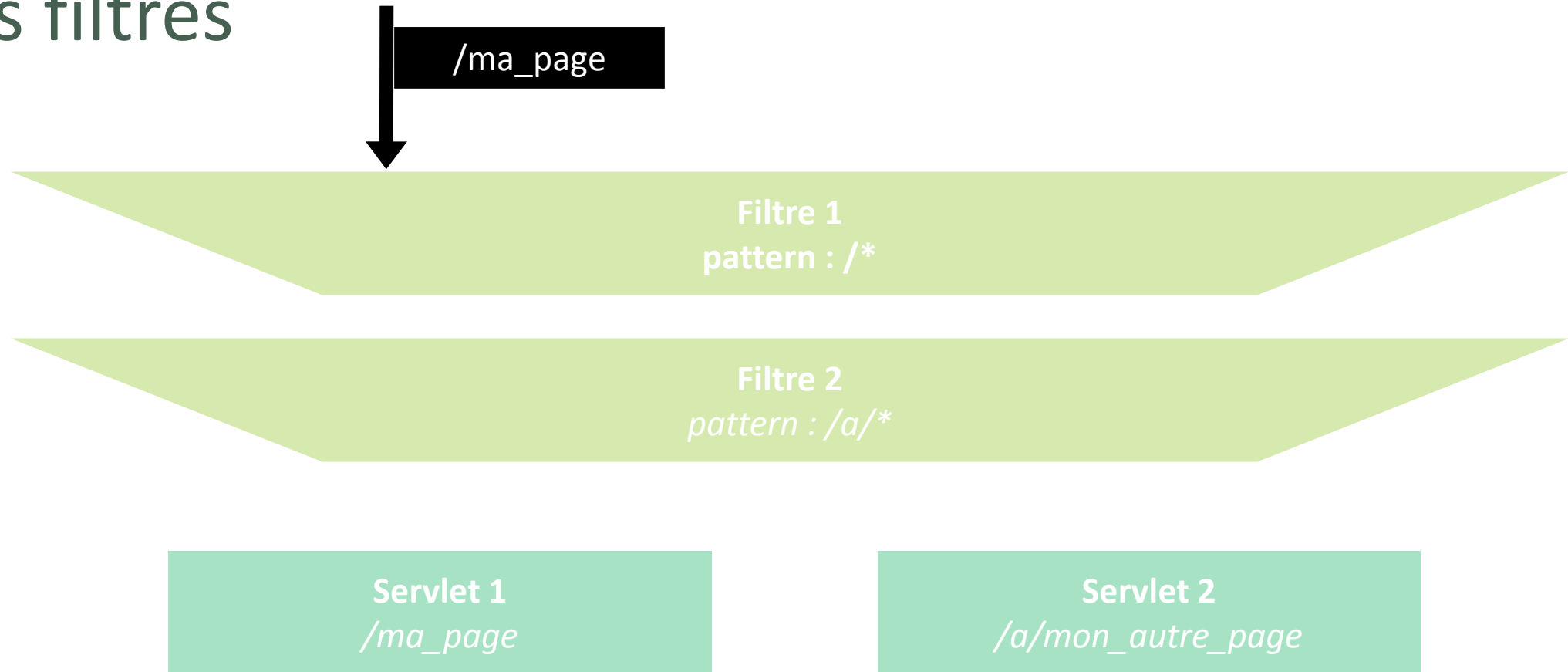
Les filtres

- Il est possible de faire passer les requêtes ainsi que les réponses à ces requêtes à travers des filtres.
- Chaque filtre est lié à un mapping d'URL.

Les filtres

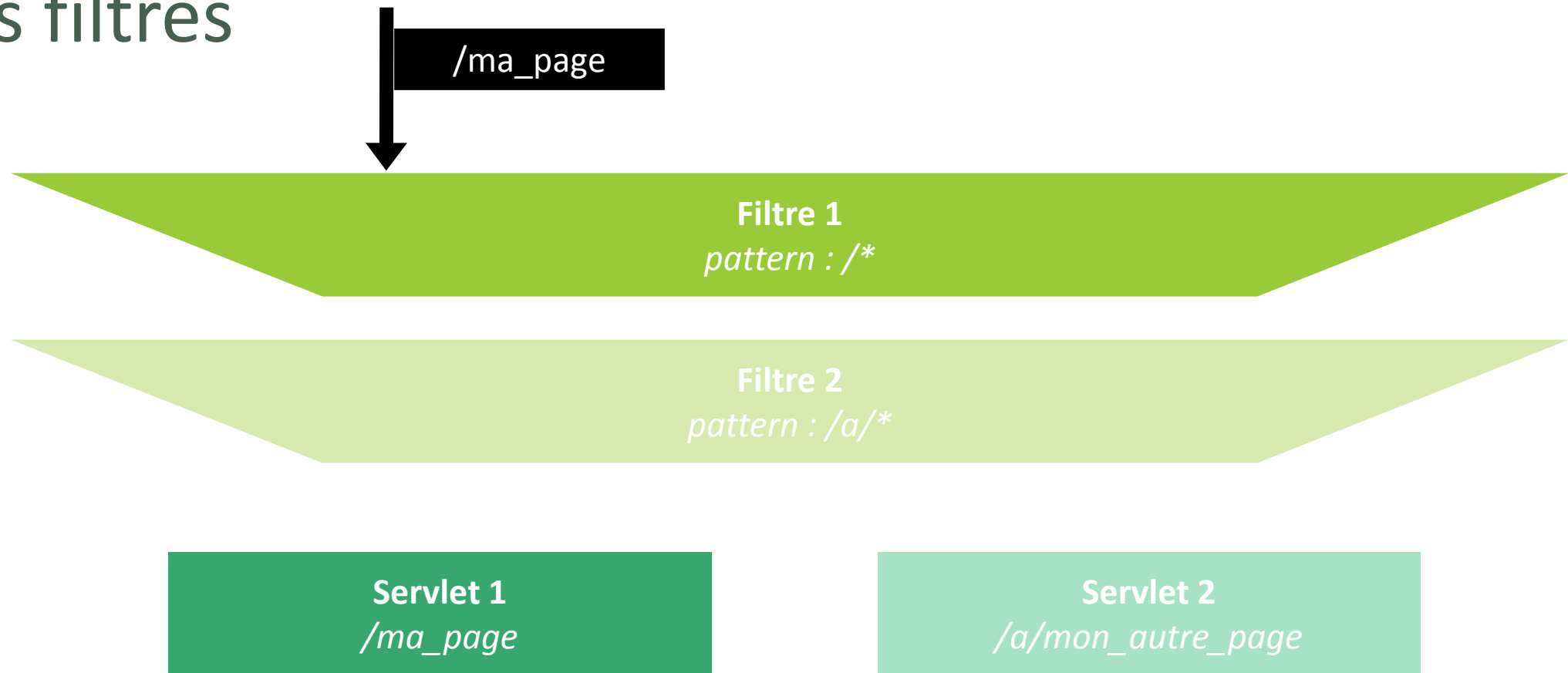


Les filtres



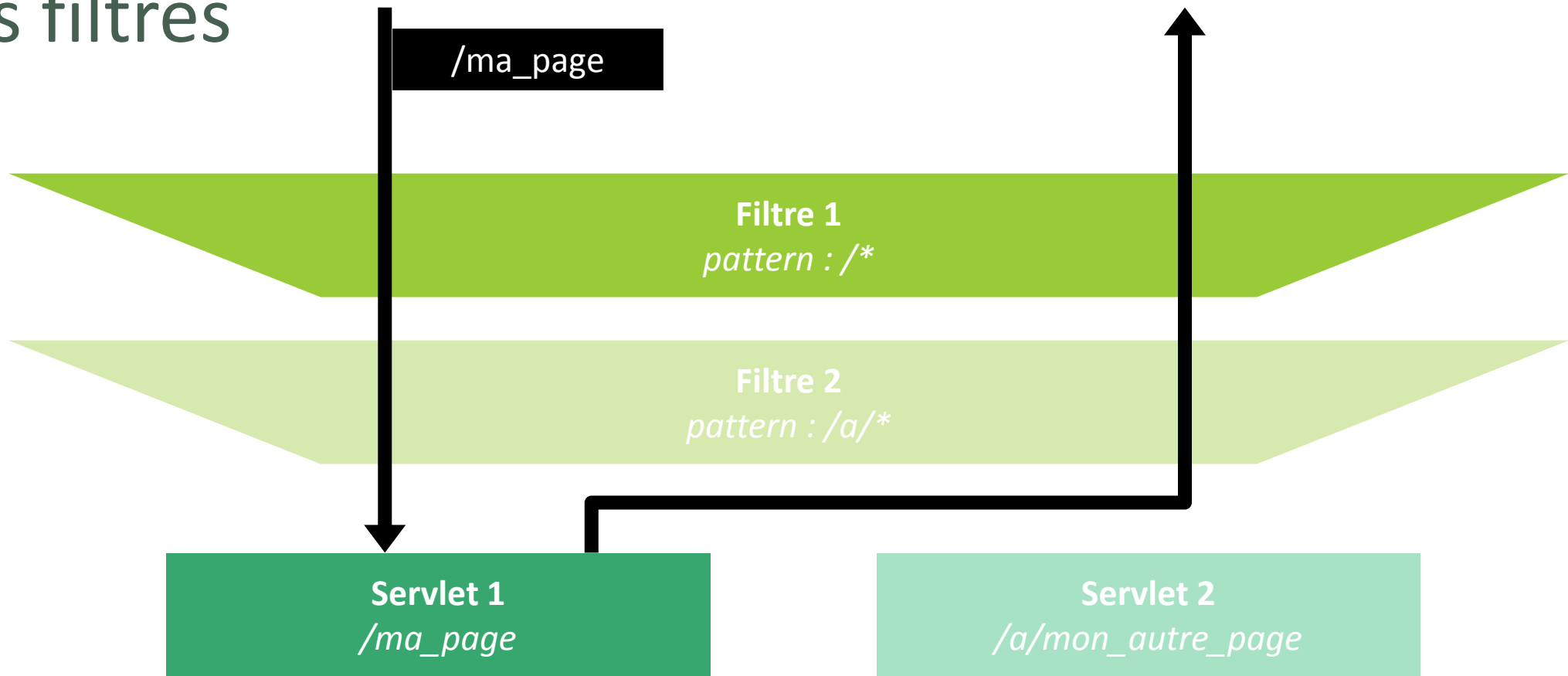
Arrivée de la requête

Les filtres



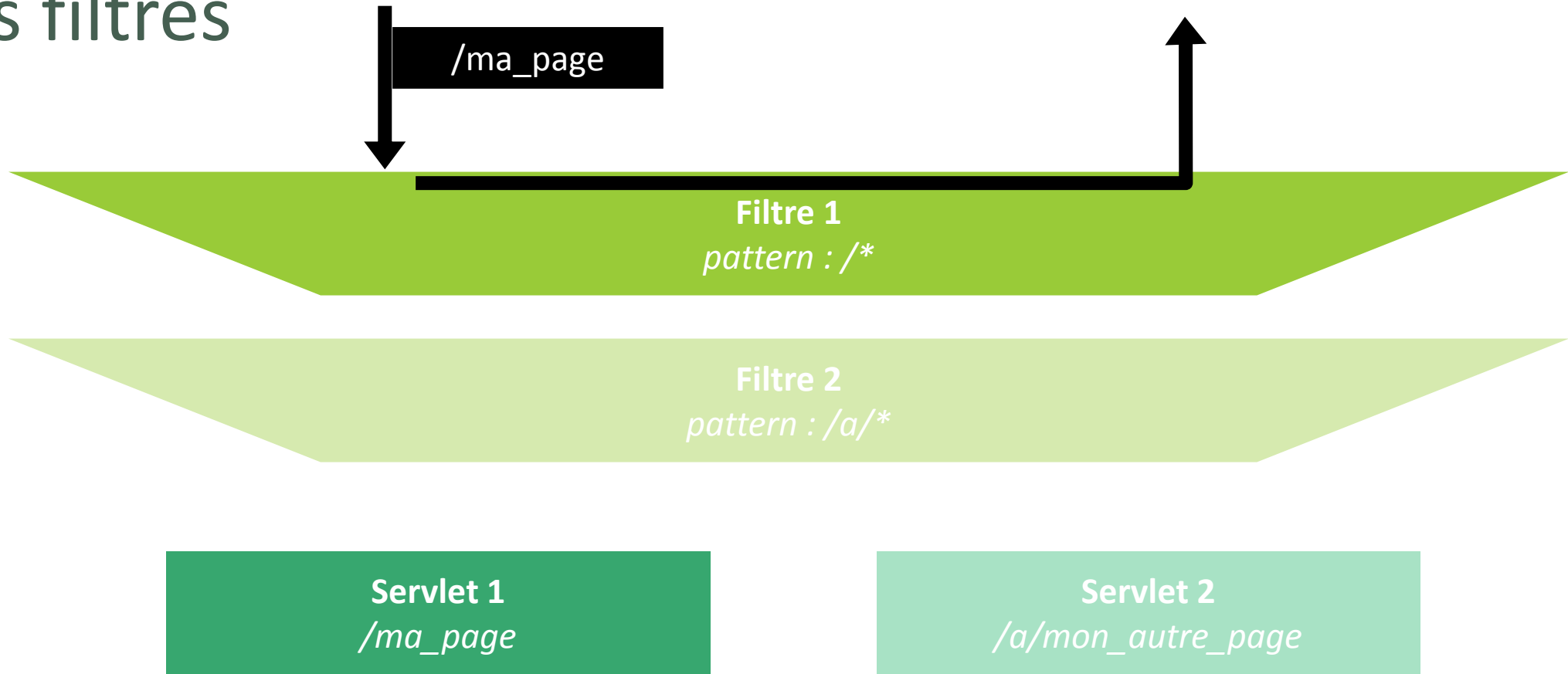
En fonction de l'url, la servlet et les filtres concernés s'activent

Les filtres



La servlet a terminé son travail, la réponse repasse dans le filtre.

Les filtres



Alternative: le filtre 1 n'a pas passé le relai à la servlet et renvoie directement la réponse

Les filtres

- Un filtre est une classe java implémentant l'interface Filter.
- La méthode principale est `doFilter()`
 - Un des paramètres de cette méthode est l'objet `FilterChain` représentant le prochain filtre ou la servlet.

```
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
throws IOException, ServletException {
    // Action faite avant de passer la main
    chain.doFilter(request, response);
    //Action faite après avoir passé la main
}
```

Les filtres

- Dans le web.xml

```
<filter>
  <filter-name>AuthenticationFilter</filter-name>
  <filter-class>my.package.AuthenticationFilter</filter-class>
</filter>
```

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

Ordre d'exécution des filtres

- L'ordre des `<filter-mapping>` dans le fichier `web.xml` donne l'ordre d'exécution des filtres
- Il est également possible de passer par l'annotation `@webFilter`
 - Aucune idée de l'ordre d'exécution dans ce cas

Bilan

- Les servlets et les filtres constituent la base de toute application web java.
- Avec, on peut donc générer des pages HTML totalement dynamiques.
- Mais ça reste quand même une belle soupe de code.
 - Peu lisible
 - Peu maintenable
 - Pénible à écrire