Environnement Tomcat

Servlets

Mise en place : NetBean

- Création d'un nouveau projet
  - Télechargez & Installer Tomcat.
  - Lancer IntelliJ
    - Nouveau projet type **Jakarta EE**, **Application web**
    - Définir le serveur de développement en indiquant le chemin vers votre serveur Tomcat
  - Analyse des fichiers créés et de l'espace de développement
    - Arborescence du répertoire
    - Classement des fichiers
    - Vues des librairies accessibles
    - Les fichiers de configuration
    - Les outils => Affichages des services, plug-in Tomcat pour IntelliJ

TD

- *Eclipse vs NetBean vs IntelliJ …*
- *Installation de tomcat*
- *Première servlet*
- *Première jsp*
- *Les objets implicites*
- *Jouez avec le conteneur*
- *Filtres*
- *Listeners*
- *Dispatcher*

Rappel des bases

- **Servlet :**
  - classe java dérivée de **HttpServlet**
  - Méthodes :
    - **Init(), Destroy()** : déclenchées par le serveur à l'instanciation de la servlet.
      Soit au premier appel sur cette servlet, soit à l'initialisaiton de l'application si c'est indiqué dans le paramétrage (load on startup)
    - **doGet / doPost(request, response)**
      Code executé lors de chaque appel vers la servlet (methode GET ou POST selon l'appel client)
  - **Objets implicites**
    - Session
    - Config
  - **Configuration** dans web.xml ou par annotations dans la classe de la servlet
- **Tomcat**
  - Serveur web et conteneur de servlet
  - Utilise le fichier web.xml de l'application web ou les annotations pour donner l'accès aux servlet
  - Possède une configuration propre prédéfinie et modifiable par ses fichiers web.xml, server.xml, context.xml
- **IntelliJ**
  - Projet application web
  - Environnement de développement
  - Contrôle le serveur tomcat pour faciliter la mise en place des servlets
  - Génère des annotations directement dans les fichiers java pour le mapping des servlets.

Première servlet

- **Ecrivez votre première servlet dans votre logiciel de développement.**

  Package : servlets à créer dans le package sources

  Nom de classe : HelloWorldServlet

  - Une fois écrite, ou plutôt générée par le logiciel,on peut paramétrer la servlet par deux manières différentes:

    - Dans le fichier web.xml du projet

    Nom de servlet : HelloWorld

    Url d'accès : ~ServletTP1/Bienvenue

    Ajouter lui un paramètre :

    - *Nom : « message »*
    - *Valeur : « paramètre de config »*

Dans IntelliJ tout ceci est déjà généré dans votre projet par défaut, le nom et l'url dans l'annotation @WebServlet

Première servlet : web.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns="https://jakarta.ee/xml/ns/jakartaee"
     xsi:schemaLocation ="https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/web-app_6_0.xsd"
     version="6.0">
     <servlet>
          <description>PifPaf</description>
          <display-name>helloServlet</display-name>
          <servlet-name>helloServlet</servlet-name>
          <servlet-class>HelloServlet</servlet-class>
          <init-param>
               <param-name>greeting</param-name>
               <param-value>Hello, World!</param-value>
          </init-param>
     </servlet>
     <servlet-mapping>
          <servlet-name>helloServlet</servlet-name>
          <url-pattern>/helloServlet</url-pattern>
     </servlet-mapping>
     <display-name>TomcatDemo</display-name>
     <welcome-file-list>
          <welcome-file>index.html</welcome-file>
          <welcome-file>index.htm</welcome-file>
          <welcome-file>index.jsp</welcome-file>
          <welcome-file>default.html</welcome-file>
          <welcome-file>default.htm</welcome-file>
          <welcome-file>default.jsp</welcome-file>
     </welcome-file-list>
</web-app>
```

Première servlet : Annotations

```
@WebServlet(name="helloServlet", value = "/hello-servlet",
initParams = @WebInitParam(name = "config", value = "myConfigParameter"))
```

Name	Type	Required	Description
value or **urlPatterns**	*String[]*	Required	Specify one or more URL patterns of the servlet. Either of attribute can be used, but not both.
name	*String*	Optional	Name of the servlet
displayName	*String*	Optional	Display name of the servlet
description	*String*	Optional	Description of the servlet
asyncSupported	*boolean*	Optional	Specify whether the servlet supports asynchronous operation mode. Default is false.
initParams	*WebInitParam[]*	Optional	Specify one or more initialization parameters of the servlet. Each parameter is specified by `@WebInitParam`annotation type.
loadOnStartup	*int*	Optional	Specify load-on-startup order of the servlet.
smallIcon	*String*	Optional	Specify name of the small icon of the servlet.
largeIcon	*String*	Optional	Specify name of the large icon of the servlet.

Première servlet

- **Analyser le code et les fichiers produits**

  Regarder la structure du code auto généré.(Répertoire work pour tomcat)

  Jouez avec l'éditeur à l'aide de **CTRL+ESPACE** pour découvrir les objets mis à disposition de notre servlet:

  - Objets passés en paramètres : **request**, **response**
  - Méthodes obtenues par la dérivation de **HttpServlet (this)**:

    getServletConfig, **getServletContext**, **getInitParameter** …

- **Elle devra ensuite répondre par un « Hello world ! » en bon et du forme.**

  Répondre au client un hello world sympa, pensez à une mise en forme HTML

  Afficher le paramètre « message » que vous avez déclaré au départ

  Déclarer un paramètre pour le **contexte d'application** et afficher le dans la console

- **Surcharger les methodes init() et destroy() pour afficher un message dans la console**
  - **Genre : servlet initialisée, servlet détruite**
  - **Mieux : Récupérez nom de la servlet pour compléter ces messages.**

- **Créez un fichier html que vous nommerez « formBonjour .html» contenant deux formulaires.**

  Chaque formulaire comportera une zone de saisie texte nommée « nom » et pointera sur la servlet

  - Le premier formulaire utilisera une method GET
  - Le second avec une method POST
  - Placez cette page en accueil du site

- **Votre servlet répondra aux formulaires en indiquant la méthode choisie (get/post)**
  - Et dans les deux cas ajouter un message de réponse.
    - **Observez les objets passés en paramètres à la servlet :**

      Récupérez la saisie des formulaires et renvoyez les dans une phrase :

      Hello « NOM_SAISI », saisi dans un des formulaires de type « TYPE(GET/POST) »

Listeners

- Créer un listener pour **le contexte d'application**
  - Créer un package « listeners »
  - Créer la Class EcouteurApplication implémentant **ServletContextListener**
  - Renseigner le fichier web.xml ou utiliser l'annotation **`@WebListener`**
- Afficher un message à la création du contexte et à sa destruction.
- optionnel : Paramétrer ces messages dans web.xml et modifier la classe pour qu'elle récupère ces informations.

Filters

- **Créer un filtre pour toutes les requêtes sur l'application**
  - Créer un package «filters »
  - Créer la Class filtreApplication implémentant Filter
  - Renseigner le fichier web.xml ou l'annotation `@WebFilter`
    - Filtre
    - Mapping

```
@WebFilter(value = "/hello-servlet",
initParams = @WebInitParam(name = "configFiltre2", value =
   "myConfigParameterFiltre2") )
```

- **Créer un second filtre, pour les requêtes sur la servlet HelloWord uniquement**
  - Réaliser les mêmes affichages que précédemment
  - Que remarquez-vous à l'affichage console ?
    - Prédire l'ordre des filtres n'est pas possible avec les annotations
  - Comment comprenez-vous l'argument : FilterChain chain
- Afficher sur la console le contenu du paramètre « nom » provenant de la requête
- Insérer dans la réponse un message provenant d'un paramètre que vous aurez spécifié dans la configuration du filtre dans le web.xml ou dans l'annotation

Beans : Classes Service ou Métier

- **Créer un nouveau package « beans » dans le répertoire sources.**
- **Créer un classe Compteur**
  - Champ : int n initialisé à zéro;
  - Getter et setter appropriés
  - Une méthode : increment() qui incrémente n
- **Créer une nouvelle servlet CompteurServlet accessible à « /Compteur »**
  - Ajouter lui un champ compteurField de type Compteur
  - À chaque requête la servlet incrémentera le compteur et affichera sa valeur
  - Expliquez ce qui est compté.
  - Gérer un compteurSession dans la methode doGet ou processRequest

    Ce compteur sera placé en session et à chaque appel, récupéré, incrémenté et afficher

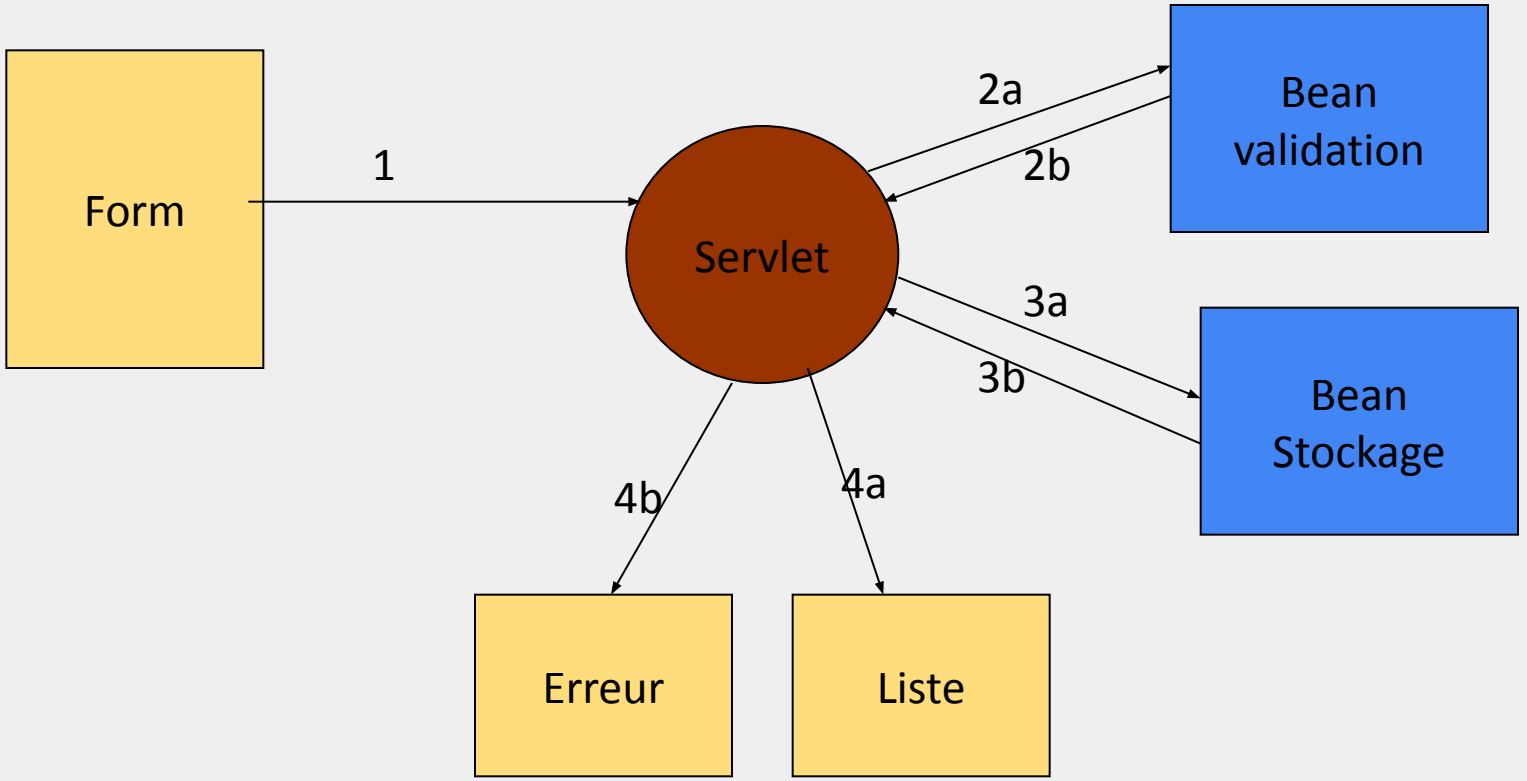
    Pensez qu'au premier appel, il n'existe pas encore.
  - Expliquez quel problème pourrait survenir avec le compteurField dans le cadre d'une application mise en ligne pour de nombreux utilisateurs.

    Comment y remédier?

Beans II

- Un bean de validation :
  - Créer une page html contenant un formulaire
    - Nom
    - Age
    - DateNaissance
    - Tel
  - Créer un bean Personne comportant les champs
    - Nom : String
    - Age : int
    - DateNaissance : Date
    - Tel : String
      - Créez les getters et setters. Pour les champs Date et Age, créez un setter supplémentaire avec un paramètre d'entrée String.
        - Le formulaire ne gère que des strings, c'est le bean qui convertira les valeurs.
  - On va créer une servlet qui va utiliser le bean pour valider les données saisies et enregistrer les Personnes

Description



Cheminement des requêtes

- À la connexion au site, l'utilisateur reçoit le formulaire de saisie d'une personne.
- Le formulaire est soumis à la servlet de contrôle, elle va diriger les opérations à réaliser sur les données :
  - Vérification de la validité des données saisies à travers un bean spécialisé.

    Ce bean comportera un champ Erreur type ArrayList<String> initialisé et modifié en un message spécifique si une erreur de format est rencontrée.
  - Redirection en fonction de l'existence d'une erreur

    vers une jsp : qui affichera l'erreur rencontrée et un lien vers le formulaire pour saisir à nouveau.
  - Traitement et Redirection s'il n'y a pas d'erreur (objet.isEmpty)

    Enregistrement de l'objet personne dans une liste qui est stockée en session (ou application).

Bean de validation

- On pourrait imaginer un objet particulier uniquement destiné à vérifier les champs du formulaire mais pour condenser un peu, on va utiliser le bean Personne.

  - Rajouter un champ ArrayList<String> messageErreur
  - Dans chaque setter de l'objet, faites votre vérification et ajouter un message lié à l'erreur dans message erreur à l'issu d'un test négatif

    Par exemple si le nom est vide, ajouter « pas de nom saisi » à messageErreur

    Dans un premier temps, vérifier que chaque champs n'est pas vide. Plus tard, on vérifiera les formats.

Bean de stockage

- On peut imaginer un bean spécifique au stockage des personnes, qui se connecterait à une base de donnée par exemple. Mais …non.
- Pour faire simple au départ, on va simplement créer une ArrayList<Personne> vide à **l'initialisation** de notre servlet de contrôle des opérations, qui la placera dans le **scope application** (contexte d'application) ou session.

Servlet de contrôle

- À vous de jouer.
- Voici les fonctions à réaliser :
  - Initialisation la liste de personnes

    Soit vide, soit avec une ou deux personnes.
  - Récupération des paramètres issus du formulaire, création/validation d'un objet personne.
  - Redirection vers la page adéquat : erreur ou liste des personnes.

Page Erreur.jsp

- Doit avoir accès au bean personne défectueux
- Affiche le ou les messages d’erreurs associés et propose un lien vers la page de saisie.

Page listePersonne.jsp

- Affiche un tableau de la liste des personnes enregistrées.
- Propose un lien vers le formulaire de saisie.

TD JSP : page 1

- Créer un nouveau projet web
  - Créer le répertoire jsp dans le répertoire web pages
  - Créer un package beans dans le répertoire source packages
- Ecrire une première page jsp nommée Header qui affiche
  - Votre nom
    - Par un scriplet : <% … %>
    - Par une expression <%= …%>
    - Par un bean simple qui comporte votre nom comme propriété.
    - Par un accès au web.xml où vous aurez placé votre nom en paramètre du contexte.
  - La date du jour
  - La date de création de la page
- Placer tout le contenu affiché dans une div avec une couleur de fond de votre choix
- Retirer ensuite tout le code html hors de votre div en choisissant aussi une seule des 4 solutions précédentes pour afficher votre nom.

TD JSP : page 2

- Créer une seconde page jsp nommée Entrée.
  - Utiliser la directive d'inclusion afin d'importer le contenu de la page header dans la nouvelle page.
  - Utiliser l'action jsp d'inclusion pour obtenir le même résultat (les deux inclusions doivent être affichées).
  - Créer un formulaire de saisie simple vous permettant de choisir une couleur

    Method : post ou get

    Action : laisser vide afin que cette même page soit rappelée

    Traitement :
    - Ecrivez un scriplet qui récupère le paramètre de couleur choisie et le place dans un attribut de session
    - Modifier le code de la page Header.jsp pour que la couleur choisie devienne celle du fond de la div définie

    Quel résultat obtenez-vous ?
    - La couleur change-t-elle immédiatement ou non ?
    - Rafraîchissez la page….la couleur s'applique-t-elle ?

    Réfléchissez au comportement et trouver une solution pour que la couleur s'applique directement

    Si c'est le cas….tant mieux…mais expliquez pourquoi ☺ !
    - Indice : emplacements et natures des codes
  - Supprimer ensuite l'une des deux inclusions faite pour avoir une page propre.

TD JSP : Page 3

- Gestion d'erreurs
  - Créer une JSP nommée Calcul.jsp
    - Ecrivez un formulaire de saisie comportant deux champs : dividende et divideur
    - Un troisième champ dessous nommé quotient que tag'rez
      - <input type="text" name=«quotient » *disabled*></input>
    - Coder une partie de traitement qui affichera le résultat de (dividende / diviseur) dans le champ quotient.
  - Créer une JSP nommée ErreurCalcul
    - Cette page sera affichée en cas d'erreur sur l'opération précédente.
    - Faite lui afficher la nature de l'erreur ainsi qu'un lien de retour au formulaire de saisie.
    - Placez les bons tags de directives associées à la gestion de l'erreur
  - Ajouter aux page l'entête Header.jsp du site
  - Ajouter un lien vers la page Calcul.jsp dans le body de la page Entrée.jsp
  - Ajouter un lien vers la page Entrée.jsp dans Header.jsp