Sommaire

[TP 0- Mise en place de l’environnement de développement Web 3](#_Toc423955693)

[TP 1- Première servlet : DateServlet 7](#_Toc423955694)

[TP 2- Cycle de vie de DateServlet 8](#_Toc423955695)

[TP 3- Login 10](#_Toc423955696)

[TP 4- Pages jsp : page principale, page d’erreur, page de sortie 11](#_Toc423955697)

[TP 5- Page d’accueil 14](#_Toc423955698)

[TP 6- Affichage des comptes courants 16](#_Toc423955699)

[TP 7- Affichage de la liste des comptes épargne et du détail d’un compte 18](#_Toc423955700)

[TP 8- Virements 21](#_Toc423955701)

[TP 9- Traitement des virements 23](#_Toc423955702)

[TP 10- Redirection 25](#_Toc423955703)

[TP 11- Mise en place d’un filtre de servlet 27](#_Toc423955704)

**Objectif :** Réaliser un mini-projet mettant en œuvre les différents concepts présents dans les serveurs d’application :

* Servlets
* Java Server Pages
* Accès aux bases de données via JDBC

**Présentation du mini-projet :**

Nous réaliserons une application de consultation et de virement de comptes bancaires en-ligne (Bankonet)

**Point de départ :**

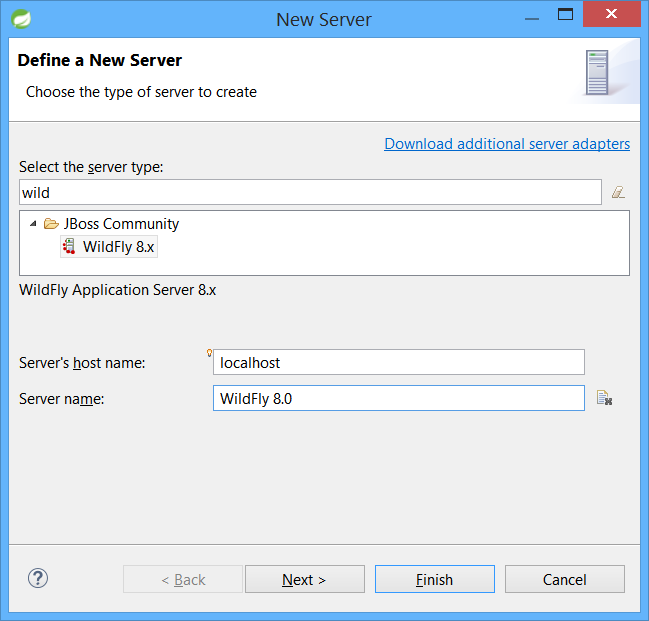
Des classes métier liées à un environnement bancaire simple, déj) développées dans le cadre de la formation Java 7 :

* Client
* Compte
* CompteCourant
* CompteEpargne
* …

# Mise en place de l’environnement de développement Web

**Installer Wildfly avec Eclipse :**

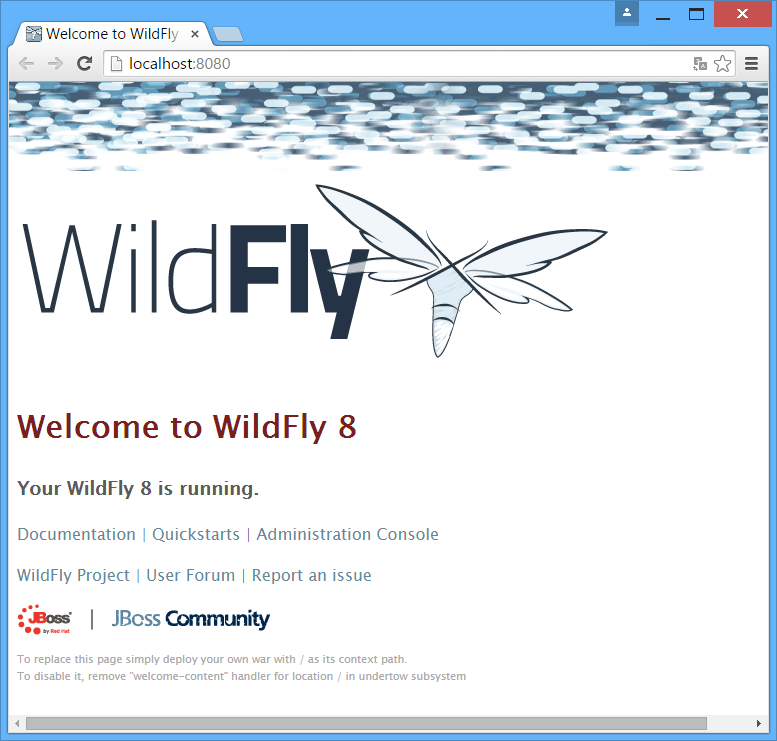
* Récupérer le serveur d’application JEE 7 « Wildfly » fourni par le formateur
* Dézipper le contenu sur votre filesystem (Ex : d:\wilfly)
* Dans Eclipse, sur la perspective « Java EE », faire un clic droit « New -> Server » dans l’onglet  « Server » pour ajouter et configurer WildFly avec Eclipse :



* Préciser le répertoire d’installation de wildfly et le JDK souhaité (JDK 8)
* Faire « Next » et « Finish »
* Le serveur apparait dans la liste des serveurs disponible : tester la bonne installation en démarrant le serveur : clic droit sur le serveur « -> Start » :

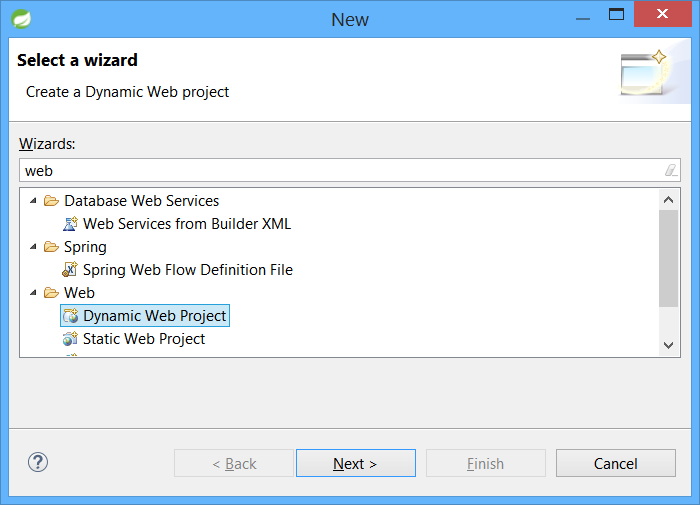


* Le serveur doit démarrer sans erreur, la console affiche les logs de démarrage de Wildfly : ouvrir un navigateur sur localhost :8080 et vérifier que tout est OK :

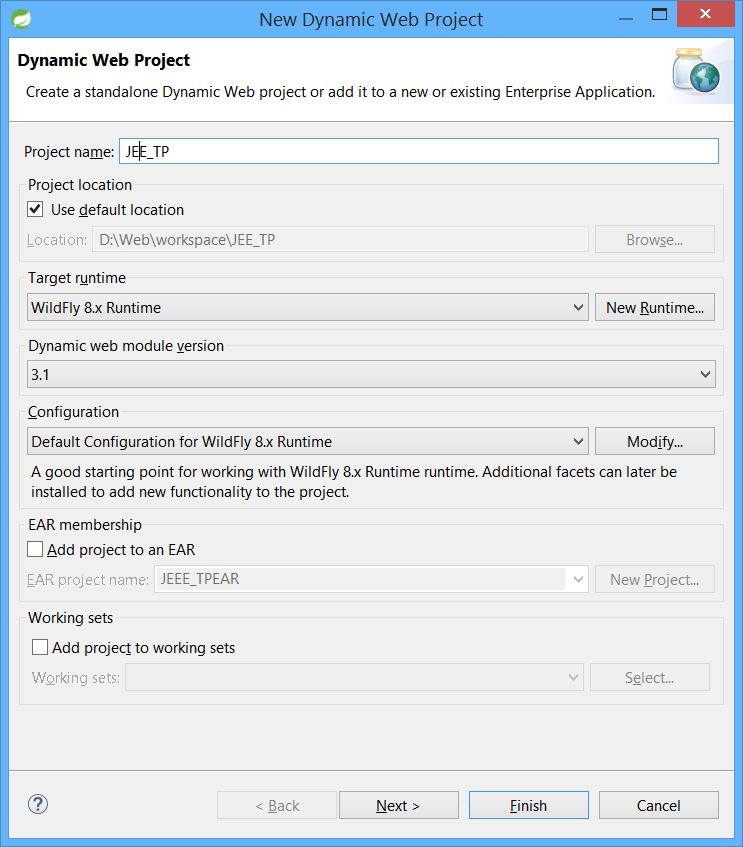


**Creation du projet « BankonetWeb » dans Eclipse :**

* Créer un nouveau projet « Dynamic Web Project » grâce à l’assistant Eclipse :



* Puis nomme le projet « BankonetWeb » et faire Next -> Next -> Finish en laissant les choix pas défaut :



# Première servlet : DateServlet

## Écrire une servlet qui affiche la date du jour dans la console.

* Dans le package com.bankonet.servlet, créer une servlet DateServlet qui hérite de la classe javax.servlet.http.HttpServlet. Dans le descripteur web.xml, elle doit être associée au mapping ‘/date’.
* Dans DateServlet, redéfinir la méthode héritée

service(HttpServletRequest req, HttpServletResponse resp)

* Implémenter la méthode service pour qu’elle affiche l’heure dans la console comme suit :

SimpleDateFormat sdf = new SimpleDateFormat("dd MMMM yyyy");

System.out.print(sdf.format(new Date())) ;

## Tester en se connectant à l’url de la servlet sur le navigateur. Il doit afficher une page blanche, et le message suivant doit apparaître sur la console :



## SystemOut O Bienvenue sur Bankonet

SystemOut O date du jour : 17 juin 2005

## 

# Cycle de vie de DateServlet

## login.jsp.

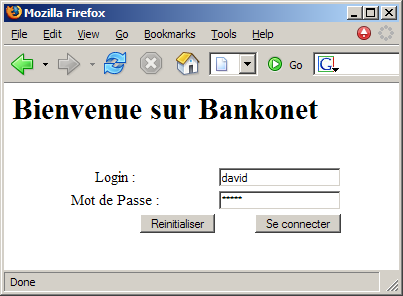
* Récupérer la page existante login.jsp
* Modifier DateServlet.service(…) pour qu’une action de type forward soit effectuée vers la page login.jsp.

this.getServletContext().getRequestDispatcher(…).forward(…);

Tester.



Résultat attendu :



## Init, destroy.

* Redéfinir la méthode DateServlet.init(ServletConfig conf). Y ajouter une trace « initialisation de DateServlet ».
* Redéfinir la méthode DateServlet.destroy().Y ajouter une trace « suppression de DateServlet ».

Tester.



## Chargement au démarrage

* Modifier le fichier web.xml pour que la servlet DateServlet soit chargée au démarrage du serveur.

Tester.



## *(question subsidiaire)* Tenter de remplacer l’action de type forward vers login.jsp par une redirection.

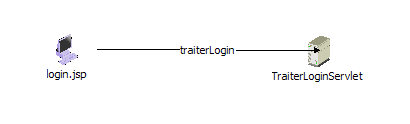
resp.sendRedirect(…) ;

Quel est le changement observé ?

Revenir à une action de type forward.

*Note : la servlet DateServlet ne sera plus utilisée par la suite.*

# Login



## TraiterLoginServlet

* Créer la servlet com.bankonet.servlet.TraiterLoginServlet. Dans le descripteur web.xml, elle doit être associée au mapping /traiterLogin.
* Dans la méthode TraiterLoginServlet.service(…),
  + Récupérer les paramètres du formulaire : identifiant et motDePasse
  + Tenter de récupérer le client associé via l’appel suivant :

BanqueService banqueService = BanqueServiceManager.getBanqueService();

Client c = banqueService.findClient(identifiant, motDePasse);

* + Si le client a été trouvé en base de données, stocker l’objet retourné dans la session. Si le client n’a pas été trouvé, une BankonetException est renvoyée (le client n’est donc jamais nul).

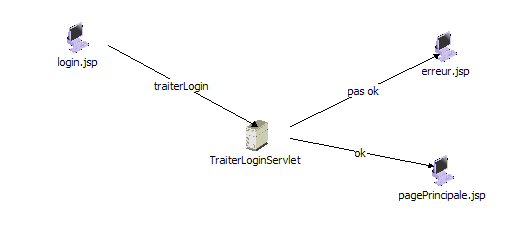
Si l’instanciation du bean Client s’est bien passée, afficher dans la console le nom et le prénom du client authentifié.

Si une exception est levée, afficher dans la console un message d’erreur du type « Connexion impossible ».

Tester.



# Pages jsp : page principale, page d’erreur, page de sortie



## Mise en place de la JSP ‘page principale’

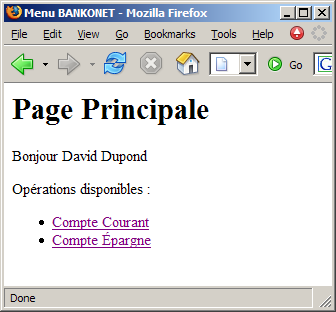
* Créer la page pagePrincipale.jsp d’après l’impression d’écran ci-dessous.
* L’objet client a été stocké précédemment dans la session (cf. LoginServlet). Le récupérer à partir de l’objet implicite session.
* Les destinations des liens ‘Compte Courant’ et ‘Compte Epargne’ ne sont pas encore implémentées. Dans un premier temps, ne pas mettre de destination pour ces liens .

<A href= "#">Compte Courant</A>

<A href= "#">Compte Epargne</A>

* Modifier la servlet TraiterLoginServlet pour qu’elle redirige la requête vers pagePrincipale.jsp en cas de réussite lors de la tentative de connexion.

Tester.



## Test sur l’affichage du lien ‘Comptes épargne’

Tout client de bankonet doit posséder au moins un compte courant. En revanche, certains clients n’ont pas de compte épargne. Dans ce cas, le lien Compte Epargne ne doit pas apparaître.

* Observer la classe com.bankonet.model.Client. Elle contient une méthode getComptesEpargne() qui renvoie une liste d’éléments.
* A partir de la javadoc du type java.util.List (Java 2 Standard Edition), chercher un moyen de tester la nullité d’une liste.
* Dans pagePrincipale.jsp, rajouter un scriptlet qui affiche le lien ‘Comptes épargne’ uniquement pour les clients qui ont un compte épargne.

Vérifier que l’utilisateur david/david a des comptes épargne. Vérifier que sandie/sandie n’en a pas.



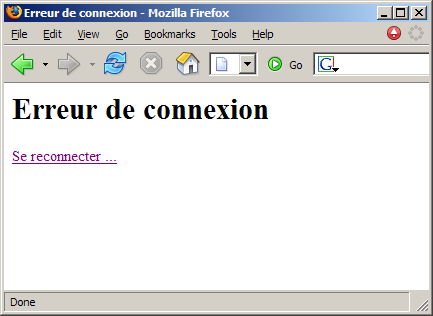
## Page d’erreur

* Créer la page erreur.jsp comme indiqué ci-dessous.
* Dans la page d’erreur, ajouter un lien hypertexte vers login.jsp. Le libellé affiché doit être "Se reconnecter".
* Modifier TraiterLoginServlet. Elle doit rediriger vers erreur.jsp en cas d’erreur de connexion.

Tester.



Résultat attendu :



# Page d’accueil

Modifier le fichier web.xml pour que la page de login du site soit login.jsp .

<web-app>

<servlet-mapping>

…

</servlet-mapping>

<welcome-file-list>

<welcome-file>login.jsp</welcome-file>

</welcome-file-list>

</web-app>

Tester en se connectant sur



**Erreur ! Référence de lien hypertexte non valide.** . La page de login doit s’afficher.

(où monPort est 8080 pour Tomcat, 9080 pour Websphere …)

## Page d’erreur par défaut

Modifier le fichier web.xml pour que la page d’erreur du site soit erreur.jsp , pour des erreurs de type ‘fichier non trouvé’ (404).

<web-app>

<servlet-mapping>

…

</servlet-mapping>

<welcome-file-list>

…

</welcome-file-list>

<error-page>

<error-code>404</error-code>

<location>/erreur.jsp</location>

</error-page>

</web-app>

Tester en se connectant sur



**Erreur ! Référence de lien hypertexte non valide.** . (où monPort est 8080 pour Tomcat, 9080 pour Websphere …)

La page d’erreur doit s’afficher.

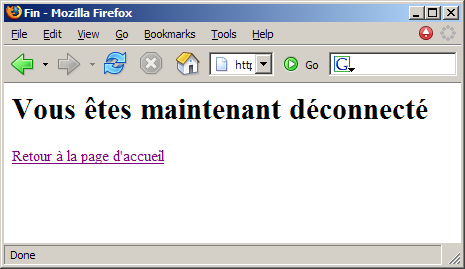
## JSP de sortie

* Concevoir la page fin.jsp contenant un message de fin d’après l’impression d’écran ci-dessous.
* Ajouter un lien hypertexte vers login.jsp ayant pour libellé "Retour à la page d’accueil".
* Ajouter une scriplet pour invalider la session (On utilisera l’objet implicite **session** de la page JSP)
* Dans pagePrincipale.jsp, ajouter un lien hypertexte vers la page de fin.

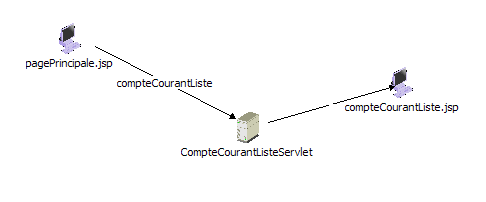
Tester la déconnexion.



Résultat attendu :



# Affichage des comptes courants



**Objectif :** Afficher la liste des comptes courants dans un tableau.

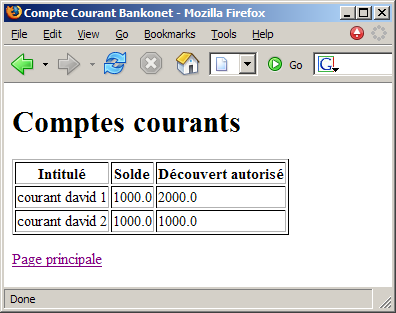
## CompteCourantServlet

* Créer la servlet com.bankonet.servlet.CompteCourantListeServlet. Dans le web.xml, elle doit être associée au mapping ‘/compteCourantListe’.
* Dans sa méthode service(…), tester l’existence d’une session. Si cette session existe, rediriger sur la JSP compteCourantListe.jsp, sinon, rediriger vers login.jsp.

## compteCourant.jsp : partie html

* Concevoir le code html de la page compteCourantListe.jsp qui affiche dans un tableau, pour chaque compte courant les attributs Intitulé, solde et montant de découvert autorisé.
* Ajouter un lien hypertexte vers /pagePrincipale .
* Dans pagePrincipale.jsp, implémenter un lien vers ‘/compteCourantListe’.

Tester l’affichage tu tableau avec des données statiques.



## compteCourantListe.jsp : partie dynamique

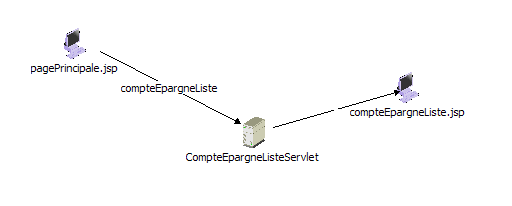
* Dans la page jsp, récupérer la liste des comptes courants du client grâce à la méthode getComptesCourants() de la classe Client.
* Parcours de la liste des comptes courants.
  + Utiliser un Iterator pour le parcours des comptes courants.
    - Récupération d’un Iterator sur une liste : maListe.iterator().
    - Pour un exemple d’utilisation d’un Iterator : se référer à la classe com.bankonet.model.Client . Elle utilise un Iterator dans sa méthode getCompte(int i).
  + Implémenter la partie dynamique de la page compteCourantListe.jsp.

Tester.



|  |  |
| --- | --- |
|  | *Il est à noter que la partie java alourdit beaucoup l’écriture et la lisibilité de la page compteCourantListe.jsp. Une syntaxe plus élégante sera proposée par la suite avec les JSTL.* |

# Affichage de la liste des comptes épargne et du détail d’un compte



**Objectif :** Utilisation des JSTL.

## Selon le même principe que le TP précédent, créer une servlet com.bankonet.servlet.CompteEpargneListeServlet et une page JSP compteEpargneListe.jsp. Utiliser les JSTL pour concevoir la page JSP. Elle doit afficher les propriétés suivantes :

* + Intitulé du compte
  + Solde du compte
  + Taux d’intérêt
  + Plafond

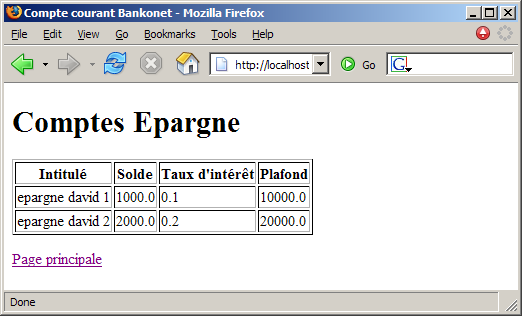
## Concevoir un lien de la page principale vers CompteEpargneListeServlet.

## Tester.



## Comparer la clarté du code entre compteCourantListe.jsp et compteEpargneListe.jsp.

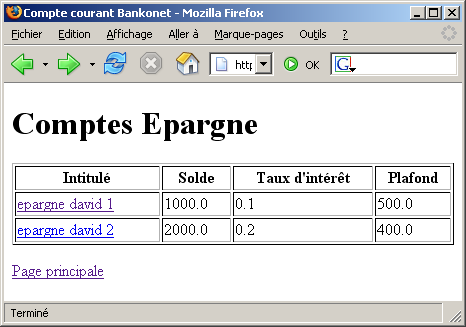
Résultat attendu :



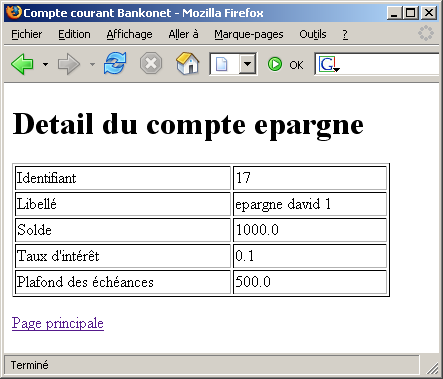
## Problématique de sélection d’un élément dans une liste

Rajouter un lien sur chacun des comptes épargnes du tableau, pour afficher le détail d’un compte. Ces liens doivent rediriger vers une servlet DetailCompteEpargneServlet . Après traitement, la servlet redirigera vers detailCompteEpargne.jsp .

* Réfléchir au moyen de récupérer l’identifiant du compte sélectionné.
* Impacter la page compteEpargneListe.jsp



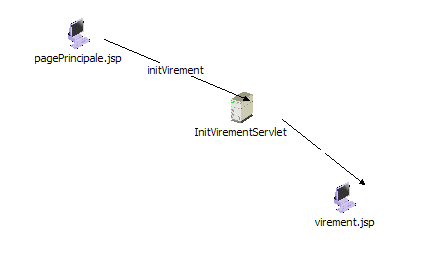
* Créer la servlet DetailCompteEpargneServlet. Y implémenter la récupération du compte choisi.
  + Récupérer le client stocké dans la session.
  + La classe Client propose une méthode qui renvoie un compte à partir d’un numéro de compte. Récupérer le compte correspondant.
  + Stocker le compte dans le contexte de requête.
* Créer la page detailCompteEpargne.jsp selon le modèle ci-dessous :



## Tester.



# Virements



**Objectif :** Mettre en place un cas d’utilisation complet ‘Virements’.

## InitVirementServlet

* Créer la servlet com.bankonet.servlet.InitVirementServlet. Dans le web.xml, elle doit être associée au mapping ‘/initVirement’.
* Dans sa méthode service(…), tester l’existence d’une session. Si cette session existe, rediriger sur la JSP virement.jsp, sinon, rediriger vers login.jsp.

## virement.jsp : partie html

* Concevoir le code html de la page virement.jsp qui affiche le compte source, le compte destination et le montant à virer. Il est conseillé de placer tous les champs dans un tableau invisible pour l’alignement.
  + Implémenter les listes déroulantes comme suit :

<select name="compteSource">

<option value="idCompte">LibelleCompte</option>

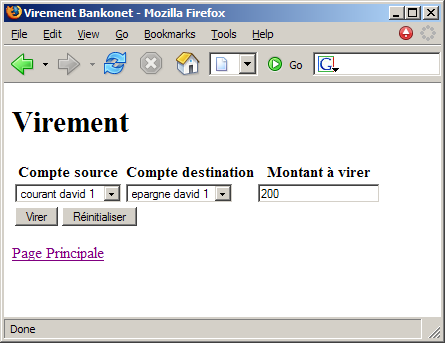
</select>

* + Quand l’utilisateur clique sur le bouton ‘Virer’, l’url /traiterVirement doit être invoquée.
* Ajouter un lien hypertexte vers /pagePrincipale .
* Dans pagePrincipale.jsp, implémenter un lien ‘Effectuer un virement’ vers ‘/initVirement’.

Tester l’affichage tu tableau avec des données statiques.



Résultat attendu :



## virement.jsp : partie dynamique

* Dans la page jsp, utiliser les JSTL pour récupérer la liste des comptes du client. Il faut invoquer la méthode qui récupère indifféremment ses comptes courants / comptes épargne (cf.méthode Client.getComptes()).
* Affichage de la liste de choix compteSource.
  + Parcourir la liste des comptes en utilisant les JSTL.
  + Générer chaque option de la liste de choix comme suit :

<select name="compteSource">

<… >  
 <option value=’<c:out value="${cpt.identifiant}"/>’>

<c:out value="${cpt.libelle} "/>

</option>  
<… >

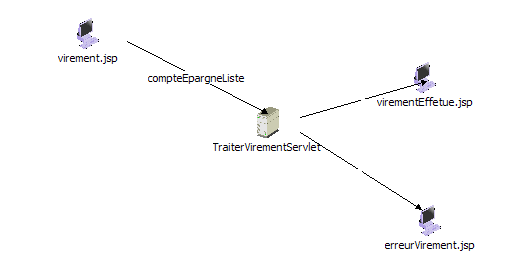
</select>

* Selon le même principe, afficher la liste de choix compteDest.

Tester l’affichage de la page dynamique.



# Traitement des virements



**Objectif :** Créer une servlet qui déclenche le virement en base de données.

## TraiterVirementServlet

* Créer la servlet com.bankonet.servlet.TraiterVirementServlet. Dans le web.xml, elle doit être associée au mapping ‘/traiterVirement’.

## Affichage des données saisies dans la console

|  |  |
| --- | --- |
|  | *Rappel : pour convertir un e chaîne de caractères en int :*  *int i = Integer.parseInt("2") ;*  *pour convertir une chaîne de caractères en float :*  *float f = Float.parseFloat("2.0") ;* |

* Récupérer les paramètres saisis par l’utilisateur dans le formulaire de virement (compteSource, compteDest, montant).
* Vérifier que le montant saisi est bien un float.
  + Tenter de le convertir en float via la méthode Float.parseFloat(…). Si la valeur saisie n’est pas de type float, une exception NumberFormatException sera lancée.
* Après la vérification du montant, afficher les paramètres saisis par l’utilisateur dans la console. Afficher également une trace si le montant saisi n’est pas un float.

Tester la servlet pour un montant valide et pour un montant non valide.



## Lancement du virement dans TraiterVirementServlet

|  |  |
| --- | --- |
|  | *Rappel : pour obtenir le message d’une exception :*  *e.getMessage() ;* |

* En observant la classe com.bankonet.servlet.LoginServlet, récupérer l’objet Client précédemment stocké dans la session.
* En observant la classe com.bankonet.model.Client, trouver un moyen de récupérer un Compte à partir d’un identifiant de compte.
* Via la classe com.bankonet.service.BanqueServiceManager, Instancier un objet de type BanqueService puis effectuer le virement.
* Cas favorable : si le virement se produit normalement, passer l’objet virement retourné dans la requête ou dans la session (identifier le contexte approprié).
* Cas d’exception : si le virement n’a pas pu s’effectuer correctement (compte source non provisionné …), une exception BankonetException est lancée. Quand elle se produit, passer un objet message contenant le message de l’exception dans la requête ou dans la session (identifier le contexte approprié).

# Redirection

* Supprimer les traces affichées précédemment.
* En cas de succès, rediriger vers /virementEffectue.jsp (à créer ultérieurement).
* En cas d’erreur, rediriger vers /erreurVirement.jsp (à créer ultérieurement).

## Virement effectué avec succès

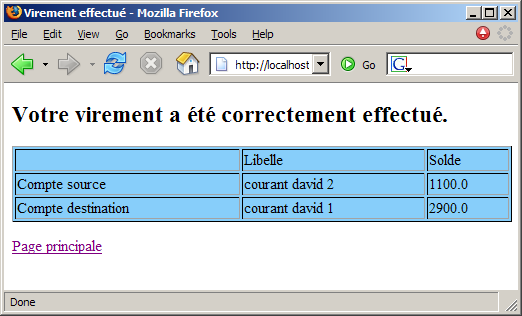
* Concevoir une JSP récapitulant le virement qui vient d’être effectué.
* Récupérer les informations à afficher via l’objet virement
* Ajouter un lien hypertexte permettant de revenir à la page principale.

Tester.



Résultat attendu :

Erreur de virement

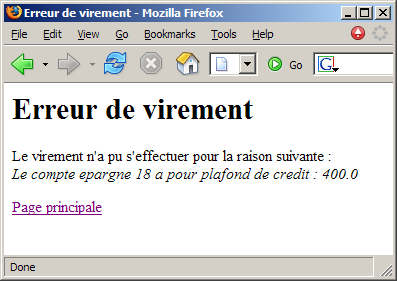


* Concevoir une JSP affichant l’erreur produite.
* Récupérer le message d’erreur via l’objet message
* Ajouter un lien hypertexte permettant de revenir à la page principale.

Tester.



Résultat attendu :



# Mise en place d’un filtre de servlet

## Dans le package com.bankonet.filter, écrire un filtre de servlet TimeFilter. Ce filtre doit afficher sur la console le temps passé dans chaque servlet.

|  |  |
| --- | --- |
|  | *Rappel : pour récupérer la date en cours (à la ms près)  :*  *System.currentTimeMillis() ;* |