TP 6 : Client léger

Pascal GRAFFION
2013/11/07 15:54

Table des matières

TP 6 : Client léger

Les servlets sont des applications Java fonctionnant du côté serveur au même titre que les CGI et les langages de script côté serveur tels que ASP ou bien PHP. Les servlets permettent donc de gérer des requêtes HTTP et de fournir au client une réponse dynamique (donc de créer des pages web dynamiques). Elles s'exécutent dans un moteur utilisé pour établir le lien entre la servlet et le serveur web. Ainsi le programmeur n'a pas à se soucier des détails techniques tels que la connexion au réseau, la mise en forme de la réponse HTTP… Les servlets sont actives (résidentes en mémoire) et prêtes à traiter les demandes des clients grâce à des threads. Les servlets étant des applications Java, elles peuvent utiliser toutes les API afin de communiquer avec des applications extérieures, se connecter à des bases de données, accéder aux entrées-sorties (fichiers par exemple)...

Le cycle de vie d'une servlet est assuré par le moteur de servlet (aussi appelé conteneur). Afin d'être à même de fournir la requête à la servlet, récupérer la réponse ou bien tout simplement démarrer/arrêter la servlet, cette dernière doit posséder une interface (un ensemble de méthodes prédéfinies) afin de suivre le cycle de vie suivant :

1. Le serveur crée un pool de threads auxquels il va pouvoir affecter chaque requête.
2. La servlet est chargée au démarrage du serveur ou lors de la première requête.
3. La servlet est instanciée par le serveur.
4. La méthode init() est invoquée par le conteneur.
5. Lors de la première requête, le conteneur crée les objets Request et Response spécifiques à la requête.
6. La méthode service() est appelée à chaque requête dans une nouvelle thread. Les objets Request et Response lui sont passés en paramètre.
7. Grâce à l'objet Request, la méthode service() va pouvoir analyser les informations en provenance du client.
8. Grâce à l'objet Response, la méthode service() va pouvoir fournir une réponse au client.
9. La méthode destroy() est appelée lors du déchargement de la servlet, c'est-à-dire lorsqu'elle n'est plus requise par le serveur. La servlet est alors signalée au ramasse miette (garbage collector).

L'objet ServletRequest encapsule la requête du client, c'est-à-dire qu'il contient l'ensemble des paramètres passés à la servlet (informations sur l'environnement du client, cookies du client, URL demandée...). L'objet ServletResponse, quant à lui, permet de renvoyer une réponse au client (envoyer des informations au navigateur). Il est ainsi possible de créer des en-têtes HTTP (headers), d'envoyer des cookies au navigateur du client, ...

Les servlets sont des classes Java implémentant des classes et des interfaces provenant des packages javax.servlet (package générique indépendant du protocole utilisé) et javax.servlet.http (package spécifique au protocole HTTP).

Hello PetStore !

Le code ci-dessous vous montre comment une servlet peut afficher le texte « Hello Petstore ! » dans une page web. d'afficher la phrase « Hello Petstore ! » et la date du jour. Comme pour la JSP et JSTL

```
import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.ServletException;

import java.io.IOException;

import java.io.PrintWriter;

import java.util.Date;


public class HelloPetstoreServlet
extends HttpServlet { // (1)


public void service(
final HttpServletRequest request, // (2)
```

```
final HttpServletResponse response)

throws IOException, ServletException {

        response.setContentType(
"text/html");

final PrintWriter out = response.getWriter(); // (3)


String param = request.getParameter(
"param"); // (4)

        out.println(
"");
        out.println(
"");
        out.println(
"<title>Hello Petstore</title>"); // (5)
        out.println(
"");
        out.println(
"");
        out.println(
"<h1><center>Hello</center></h1>");
        out.println(
"<h2><center>" + param +
"</center></h2>"); //  (6)
        out.println(
new Date()); :: (7)
        out.println(
"");
        out.println(
"");
        out.close();
    }
}
```

Une servlet doit hériter de la classe HttpServlet (1) et surcharger la méthode service (2). Ensuite, à l'aide d'un PrintWriter (3), elle peut écrire du code HTML statique (5) ou des données dynamiques (7). La servlet affiche le paramètre (6) qui lui est passé dans l'URL (4).

Cette servlet doit ensuite être packagée dans un fichier .war accompagné du fichier web.xml ci-dessous. Ce fichier permet de définir la servlet et l'alias par lequel elle sera appelée.

```
<?xml version=
"1.0" encoding=
"UTF-8"?>

<web-app xmlns=
"http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation=
"http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app&#95;2&#95;4.xsd"
    version=
"2.4">

    <display-name>Web Hello Petstore</display-name>

    <!-- Servlet Configuration -->
    <servlet>
        <servlet-name>HelloPetstore</servlet-name>
        <display-name>HelloPetstoreServlet</display-name>
        <servlet-class>HelloPetstoreServlet</servlet-class>
    </servlet>

    <!-- Servlet Mapping -->
```

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

</web-app>
```

Définissez la variable d'environnement TOMCAT_HOME. Exemple :

```
set TOMCAT_HOME=C:/Applications/java/Tomcat6.0
```

Déplacez vous dans le le répertoire Hello/src/java et utilisez les différentes tâches Ant du fichier build.xml :

1. ant war : pour compiler et packager la servlet dans le fichier hellops.war
2. ant deploy : pour déployer l'application hellops.war dans Tomcat.

Une fois ce fichier déployé dans Tomcat (et Tomcat démarré!), utilisez votre navigateur pour vous rendre à l'url http://localhost:8080/hellops/hello. Cette URL ne contient pas de paramètre, vous aurez donc le texte « null » qui apparaîtra sur la page. Par contre, en allant sur http://localhost:8080/hellops/hello?param=PetStore! la page suivante s'affiche :

Figure 1 - Exécution de la servlet

Outils

Comme nous l'avons vu plus haut, les servlets ont besoin d'un moteur pour pouvoir fonctionner. Nous utiliserons le serveur open source Tomcat pour pouvoir les exécuter. Pour l'installer, c'est très simple, il suffit simplement de décompresser l'archive dans un répertoire.

La variable d'environnement JAVA_HOME a déjà dû être configurée et désigne le répertoire d'installation du JDK. Il vous faut maintenant définir la variable d'environnement TOMCAT_HOME qui doit désigner le répertoire d'installation de Tomcat.

Pour démarrer le serveur web, rendez vous dans le répertoire %TOMCAT_HOME%/bin et exécutez le programme startup.bat. Une fenêtre telle que celle-ci devrait apparaître. Cette mise en route peut prendre quelques secondes.

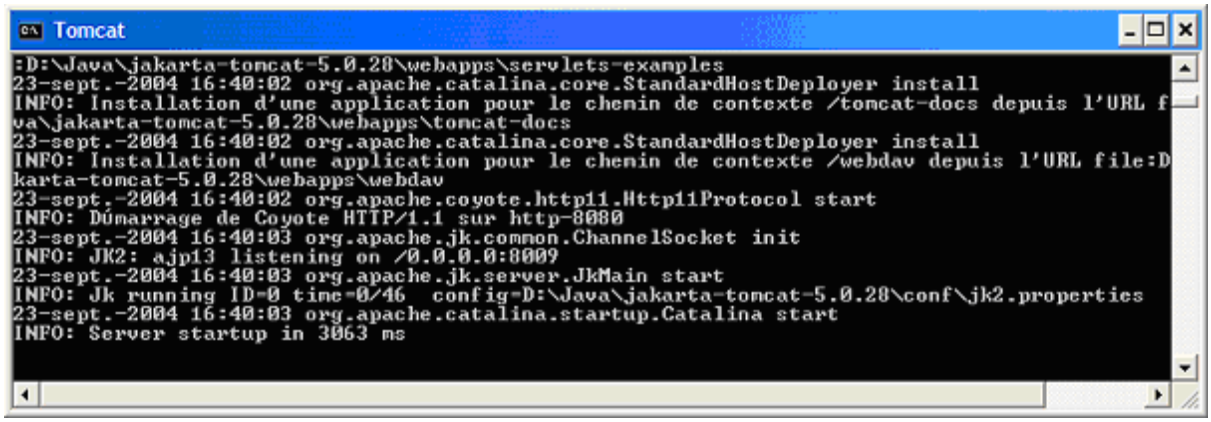


Figure 2 - Le serveur Tomcat est démarré

Une fois le message « Server startup » affiché, ouvrez un navigateur et tapez l'URL suivante : http://localhost:8080/. Si une fenêtre telle que celle qui suit apparaît, votre installation de Tomcat est terminée.

Figure 3 - Page d'accueil de Tomcat

Pour consulter la documentation Tomcat, rendez vous à l'adresse http://localhost:8080/docs/.

Ensuite, lorsque vous aurez besoin de déployer vos applications web, il vous suffira de copier vos fichiers .war dans le répertoire %TOMCAT_HOME%/webapps. Pour arrêter le serveur, exécuter le script %TOMCAT_HOME%/bin/shutdown.bat.

Nous continuons à utiliser JUnit pour exécuter les tests de recette utilisateur. Par contre, pour les applications web, il vous faut installer HttpUnit. Pour cela il vous suffit de télécharger le fichier d'installation et de le décompresser dans un répertoire que l'on référencera par une variable d'environnement HTTPUNIT_HOME.

Expression des besoins

La société YAPS entame un tournant décisif dans son activité. Depuis longtemps elle cherche à se développer dans d'autres états. Elle n'a pas encore la force économique pour créer des succursales, mais elle vient de trouver des contacts intéressants

pour ouvrir des franchises. Ces franchises sont pour la plupart des sociétés déjà existantes qui pourront vendre des animaux domestiques sous le nom de YAPS. Elles n'appartiennent pas juridiquement à la société-mère, mais peuvent utiliser son catalogue et son fichier de clients.

Ces sociétés ne sont pas toutes informatisées. La plupart continueront donc à fonctionner par le biais de fax. Les franchisés faxeront les nouvelles commandes ainsi que les informations des nouveaux clients. YAPS se doit donc d'embaucher de nouvelles personnes pour saisir informatiquement ces commandes et ces clients.

Par contre, certaines franchises possèdent des ordinateurs reliés au réseau internet. Celles-ci souhaitent pouvoir saisir les coordonnées de leurs clients au travers d'un site évitant ainsi une double saisie. Internet se développant aussi chez leurs clients, les franchisés demandent à pouvoir saisir l'adresse électronique de leurs clients.

De plus, après une étude sur son portefeuille de clients, YAPS se rend compte qu'ils sont de plus de plus nombreux à acheter régulièrement par carte bancaire. Après une enquête chez certains d'eux, il s'avère que ces clients ne sont pas réticents à l'idée que YAPS puisse stocker leur numéro de carte bancaire dans son système. Cela facilitera ainsi le passage de commandes des clients fréquents qui achètent en grosse quantité.

YAPS doit embaucher plus de personnel pour saisir les informations arrivant par fax des franchisés. Pour être plus productif elle décide d'abandonner les interfaces en mode texte utilisé par Bill et John et de n'utiliser que des interfaces graphiques.

Vue utilisateur

Diagramme de cas d'utilisation

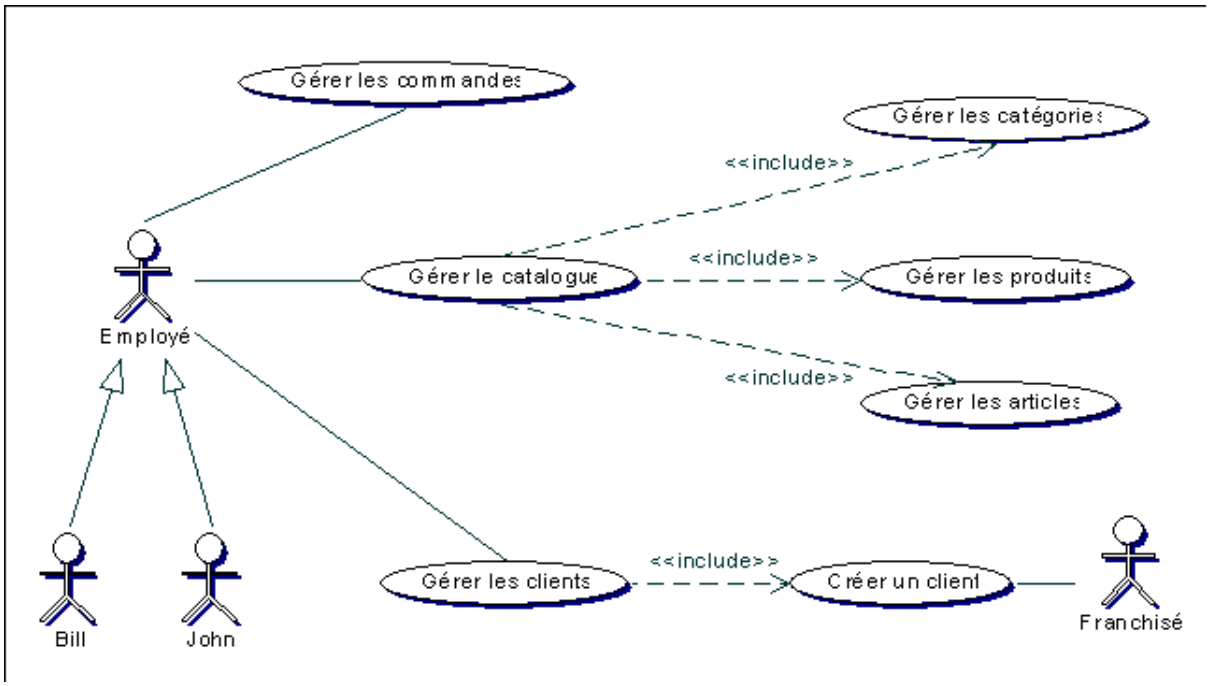


Figure 4 - Diagramme de cas d'utilisation

Les acteurs Bill et John n'ont plus de rôle particulier puisqu'ils effectuent les mêmes tâches que tous les autres employés. Ils sont représentés dans le diagramme avec une relation d'héritage sur employé. Cette relation a le même sens que dans le langage Java : Bill et John héritent de tous les cas d'utilisation de l'acteur employé.

Cas d'utilisation « Créer un client »

Nom	**Créer un client.**
Résumé	Permet de saisir les coordonnées des clients.
Acteurs	Employé et Franchisé.
Pré-conditions	Aucune.
Description	Les employés de YAPS ainsi que les franchisés veulent saisir les coordonnées de leurs clients. Les données sont les suivantes :

	- Customer Id : identifiant unique du client ((1)). Cet identifiant est construit manuellement par YAPS à partir du nom de famille suivi d'un numéro. - First Name : prénom - Last Name : nom de famille - Telephone : numéro de téléphone où l'on peut joindre le client - **Email** : adresse mail du client - Street 1 et Street 2 : ces deux zones permettent de saisir l'adresse du client. - City : ville de résidence - State : état de résidence (uniquement pour les clients américains) - Zipcode : code postal - Country : pays de résidence - **Credit Card Number** : numéro de carte bancaire du client - **Credit Card Type** : type de la carte bancaire (Visa, MasterCard...) - **Credit Card Expiry Date** : date d'expiration de la carte bancaire. Le format de cette date est MM/AA, c'est à dire 2 chiffres pour le mois et deux pour l'année séparés par le caractère '/'. Seuls les champs Customer Id, First Name et Last Name sont obligatoires ((2)).
Exceptions	((1)) Si l'identifiant saisi existe déjà dans le système, une exception est levée et l'utilisateur en est informé. ((2)) Une exception est levée si l'un des champs est manquant. ((GLOBAL)) Si une erreur se produit, une exception doit être levée.
Post-conditions	Un client est créé.

Dans le use case ci-dessus, les nouveautés sont affichées en gras. Les autres cas d'utilisations (Gérer le catalogue et gérer les commandes) sont inchangés.

Ecrans

L'interface graphique, Petstore Client, doit être enrichie pour permettre la gestion des catégories, produits, articles et clients. Pour cela, il faudra créer un menu Referential avec les sous menus permettant la gestion du catalogue et des clients.

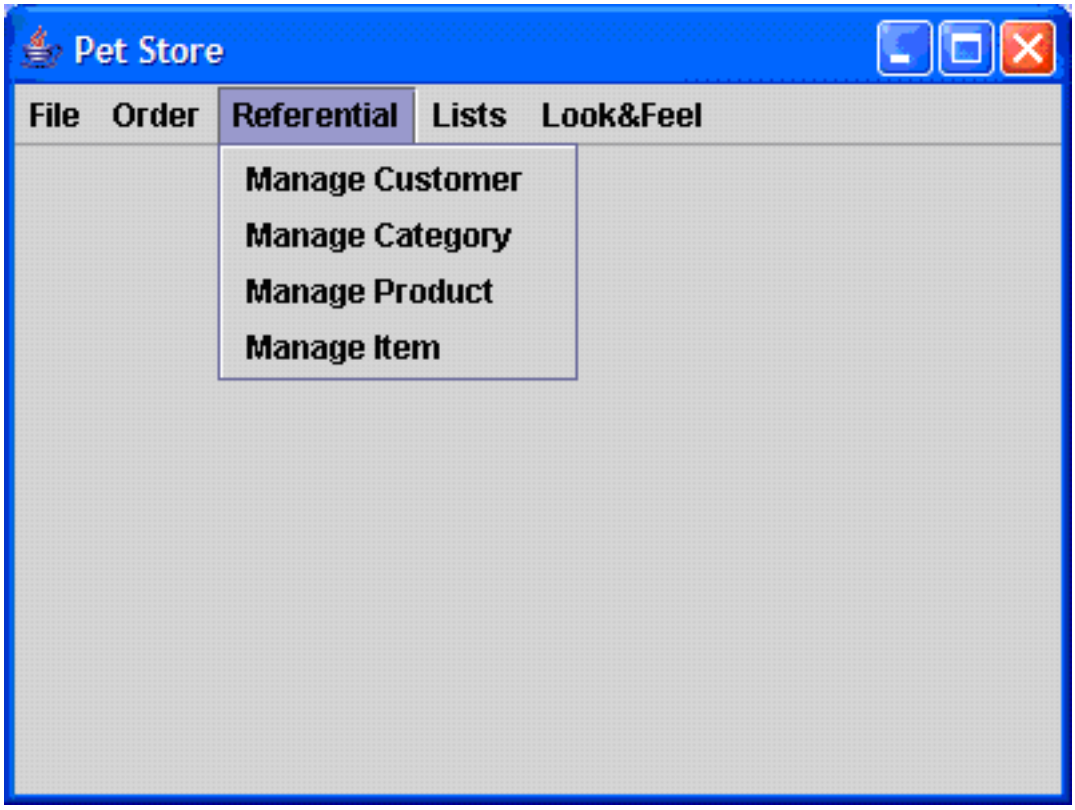


Figure 5 - Menu principal

Un clic sur chaque sous menu fera apparaître un nouvel écran. Chaque écran est constitué de la même manière : un titre, des champs de saisie et une barre de contrôle. Cette barre contient six boutons sur lesquels l'employé pourra actionner. Lorsqu'un employé saisit, par exemple, l'identifiant (1) d'un produit et clique sur Find (2), le système ramène toutes ses données. Les boutons Create, Delete et Update permettent respectivement de créer, supprimer et modifier le produit. Le bouton Reset (3) vide tous les champs de saisie et Close (4) ferme l'écran.

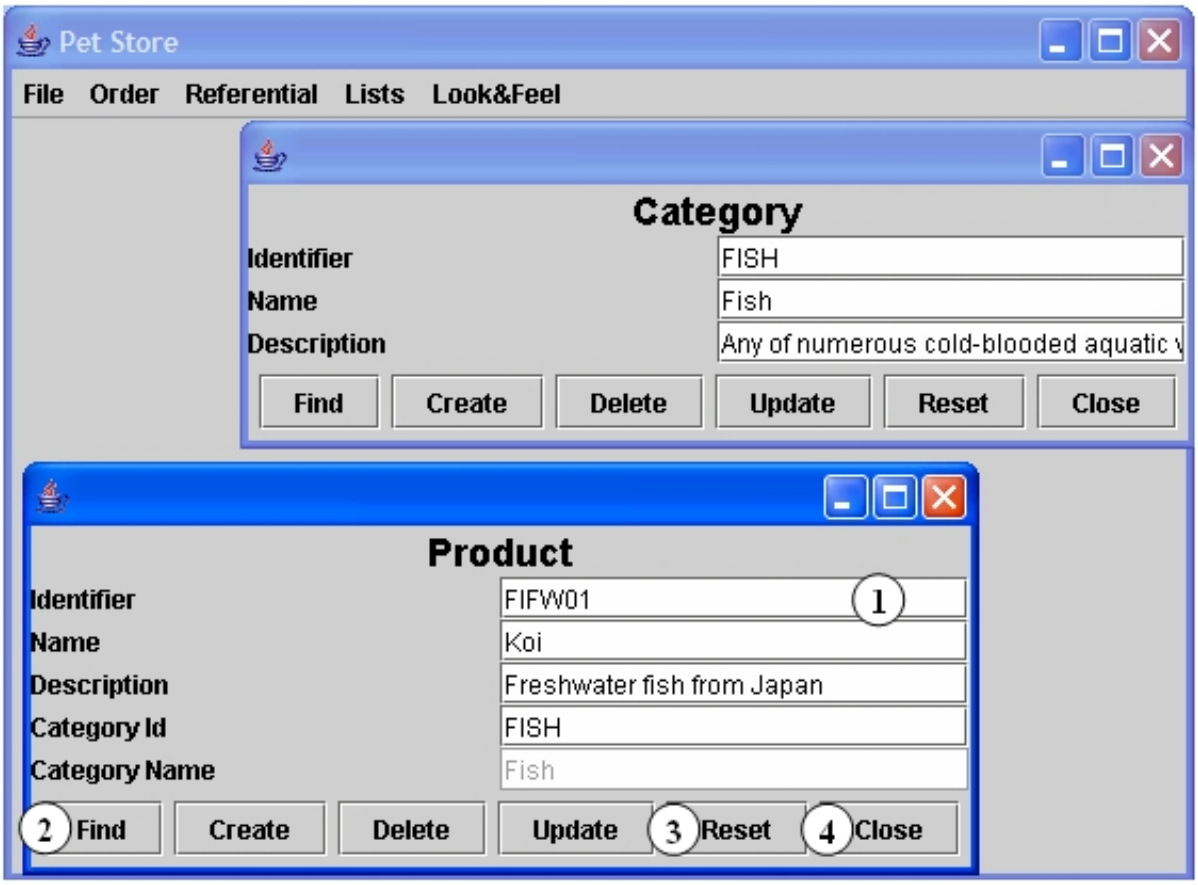


Figure 6 - Ecrans catégorie et produit

Les franchisés informatisés utiliseront l'application Petstore Web grâce à laquelle ils pourront créer de nouveaux clients à partir de la page web de la figure 7. Il leur suffira de saisir les coordonnées des clients et de cliquer sur le bouton Submit.

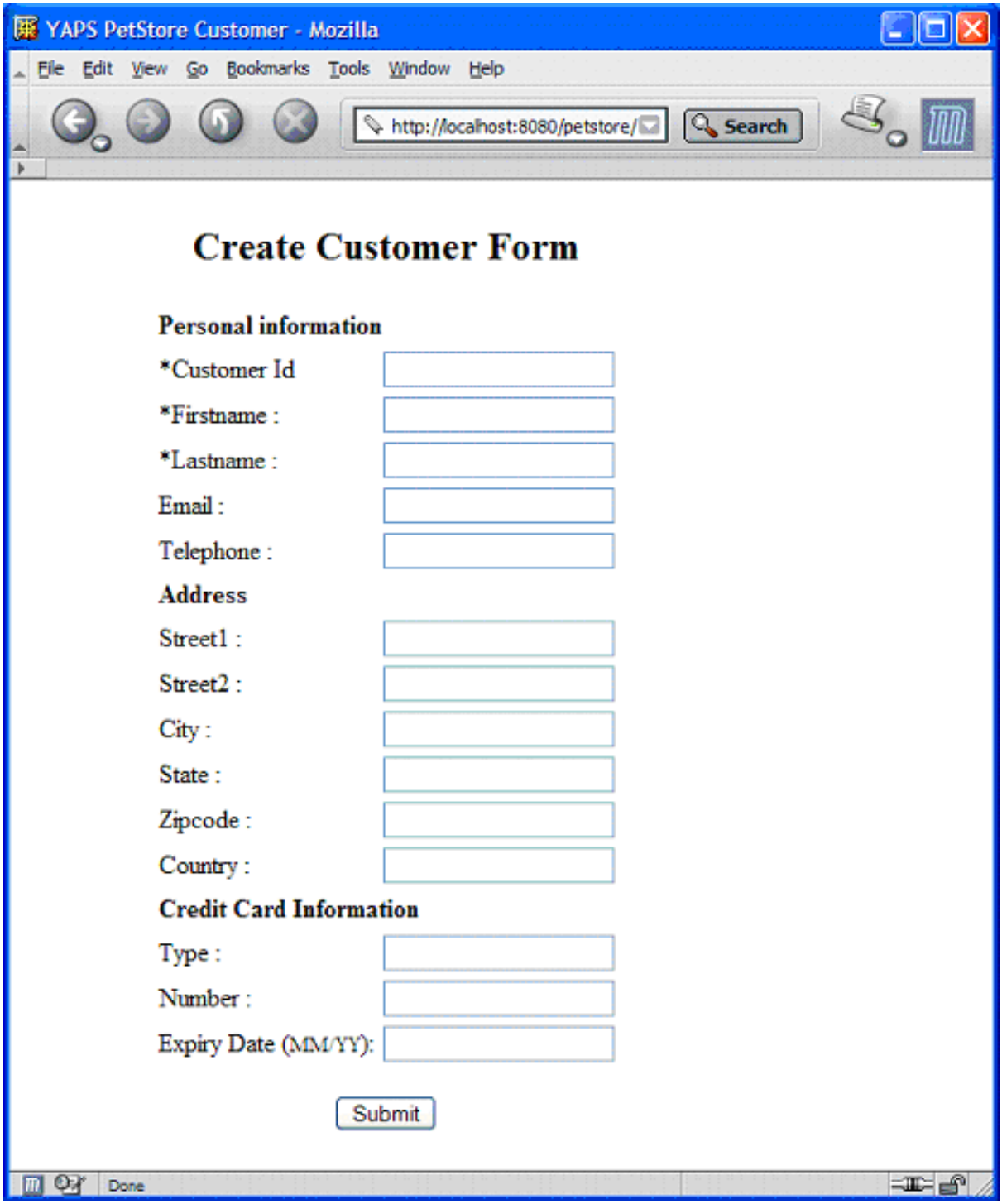


Figure 7 - Page web de saisie de client (createcustomer.html)

Si la création ne peut pas se faire, une page web d'erreur s'affiche en précisant l'exception rencontrée :

Figure 8 - Page d'erreur de création

Si au contraire la création s'effectue, un message informe le franchisé que l'opération s'est déroulée sans problème (Customer Created !!).

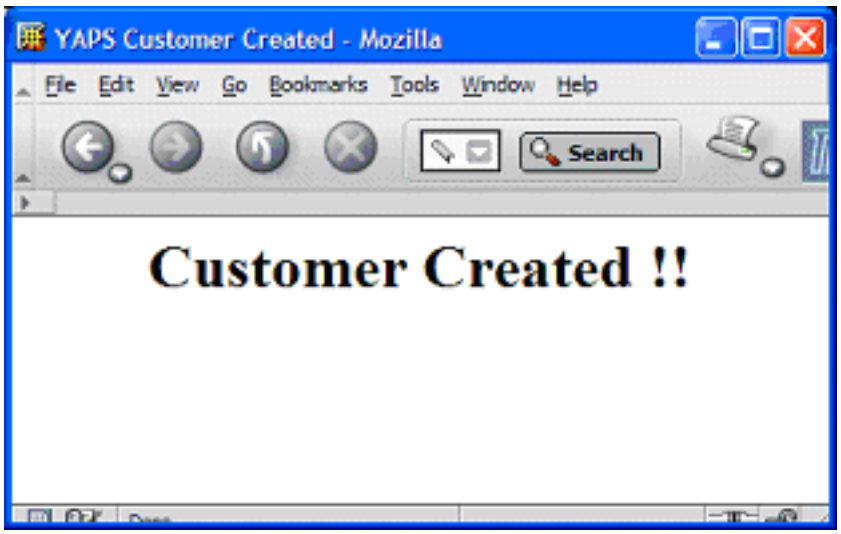


Figure 9 - Page affichée si la création a réussi

Analyse et conception

Vue logique

Les utilisateurs ont demandé à ce qu'ils puissent gérer les informations bancaires du client. Pour répondre à ce nouveau besoin, il suffirait de créer trois nouveaux attributs dans la classe Customer (credit card number, credit card type et credit card expiry date). Il s'avère que ces attributs sont exactement les mêmes et utilisés de la même façon que ceux de la classe Order. Le même cas se produit pour les attributs de l'adresse du client et de l'adresse de facturation du bon de commande. Au lieu de dupliquer ces attributs, on peut créer deux nouvelles classes : Address et CreditCard.

Les classes Address et CreditCard ont été créées par l'extraction des attributs communs des classes Customer et Order (le même procédé peut être utilisé pour les DTO avec la création des classes AddressDTO et CreditCardDTO). Cette factorisation de code utilise le remaniement Extraire Classe (Extract Class) : Vous avez une classe qui se retrouve à faire le traitement de deux classes => déplacer les attributs et méthodes dans une nouvelle classe que vous lierez à la précédente.

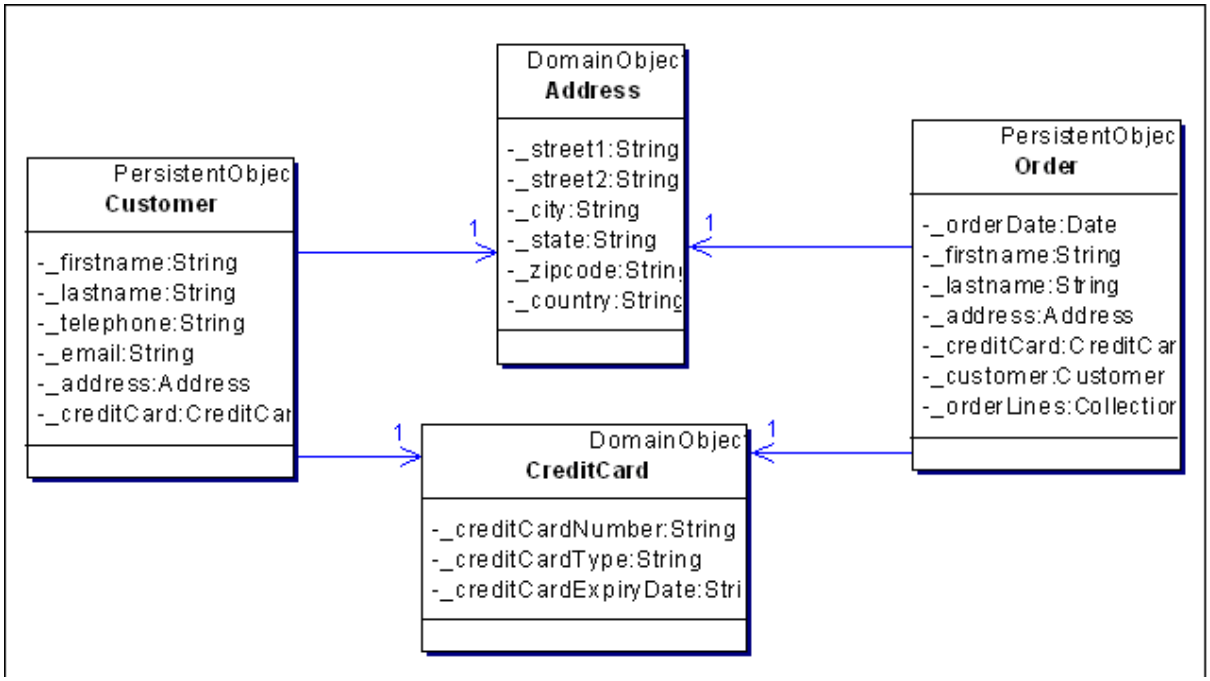


Figure 10 - Les nouvelles classes Address et CreditCard

Comme on peut le voir ci-dessus, la classe Customer possède un attribut Address et CreditCard. En faisant ce choix de modélisation, on peut avoir différents types d'accès aux données. Prenons l'exemple de la classe Customer. Pour obtenir le pays d'un client, on peut utiliser deux manières différentes d'accéder à cet attribut :

```
customer.getAddress().getCountry()
customer.getCountry()
```

L'avantage de cette dernière solution, c'est qu'on n'a pas à modifier le code des classes utilisant Customer. C'est le remaniement « Cacher Délégation ».

La clé de l'encapsulation est de ne pas rendre les attributs d'une classe publique et de créer des accesseurs (getters et setters) Par contre, ces attributs restent toujours visibles à l'intérieur de la classe. Avec le remaniement « Auto Encapsulation d'Attribut » (Self Encapsulate Field) on force la classe elle-même à utiliser ses accesseurs et non les attributs. Le remaniement « Cacher Délégation » (Hide Delegate) quant à lui, permet à une classe (Customer) de cacher le fait qu'elle utilise une autre classe (Address) en utilisant les accesseurs (getCountry() au lieu de getAddress()).

Le même problème intervient dans la classe elle-même. Par exemple, la méthode toString() de Customer utilise directement les attributs de la classe et non les getters. Le remaniement « Auto Encapsulation d'Attribut » nous dit qu'il vaut toujours mieux utiliser les accesseurs (getters et setters) plutôt que les attributs de l'objet directement.

```
public
final class Customer
extends DomainObject {


private
String _firstname;

private
String _email;

private
final Address _address =
new Address();


public
String getCountry() {
```

```
return _address.getCountry();
    }


public void setCountry(
final
String country) {
        _address.setCountry(country);
    }


public
String toString() {

final
StringBuffer buf =
new
StringBuffer();
        buf.append(
"Customer{");
        buf.append(
",firstname=").append(getFirstname());
        buf.append(
",email=").append(getEmail());
        buf.append(
",country=").append(getCountry());
        buf.append('}');

return buf.toString();
    }
```

Avant de distribuer l'application entre client et serveur, la totalité des classes était déployée sur chaque poste utilisateur. L'avantage de cette architecture est qu'il n'existe aucun problème de concurrence ni de partage des données. Prenons la classe UniqueIdGenerator. Le rôle de celle-ci est de fournir un numéro unique pour les commandes et les lignes de commandes. Maintenant que l'application est distribuée, UniqueIdGenerator ne se trouve plus sur chaque poste utilisateur mais sur le serveur qui peut être sollicité par plusieurs clients à la fois. Le fonctionnement du moteur de servlets fait que chaque appel à une méthode est géré par un thread, ce qui signifie que plusieurs appels sur une même classe engendreront la création de plusieurs instances de celle-ci. Dans le cas du UniqueIdGenerator, plusieurs instances pourraient donner le même numéro unique à différents appels. Pour éviter qu'une classe ne puisse créer plus d'une instance on peut utiliser le design pattern Singleton. Pour sécuriser la création unique d'identifiant, on appliquera ce design pattern à la classe UniqueIdGenerator.

Un singleton est un objet global pour lequel une et une seule instance de celui-ci existe dans toute l'application. Toutes les instances d'objets de ce type que vous créez font en réalité toutes référence au même objet en mémoire. Par conséquent, un singleton n'est stocké qu'une seule et unique fois en mémoire.
La création d'un singleton réside dans la prévention de la création d'un objet autre que celui que vous fournissez, c'est-à-dire une seule et même instance pour toute l'application. Pour ce faire, il suffit dans un premier temps de déclarer tous les constructeurs comme étant privé et d'avoir une méthode publique et statique permettant de retourner l'instance unique du singleton.

Vue processus

Pour créer un nouveau client, les franchisés informatisés passent par la page HTML createcustomer.html. Après avoir saisie les données, ils cliquent sur le bouton submit ce qui engendre un appel HTTP à la servlet CreateCustomerServlet. Cette dernière récupère tous les paramètres de la requête HTTP et en constitue un objet CustomerDTO. Ensuite, tout comme l'interface graphique Swing, elle appelle la méthode createCustomer de l'objet CustomerDelegate. L'appel à cette méthode se fait dans un block try/catch. Si une exception est remontée, alors la CreateCustomerServlet passera (forward) le message de l'exception à la ErrorServlet, sinon, elle affiche un message confirmant la création. Le forward est fait via la classe RequestDispatcher.

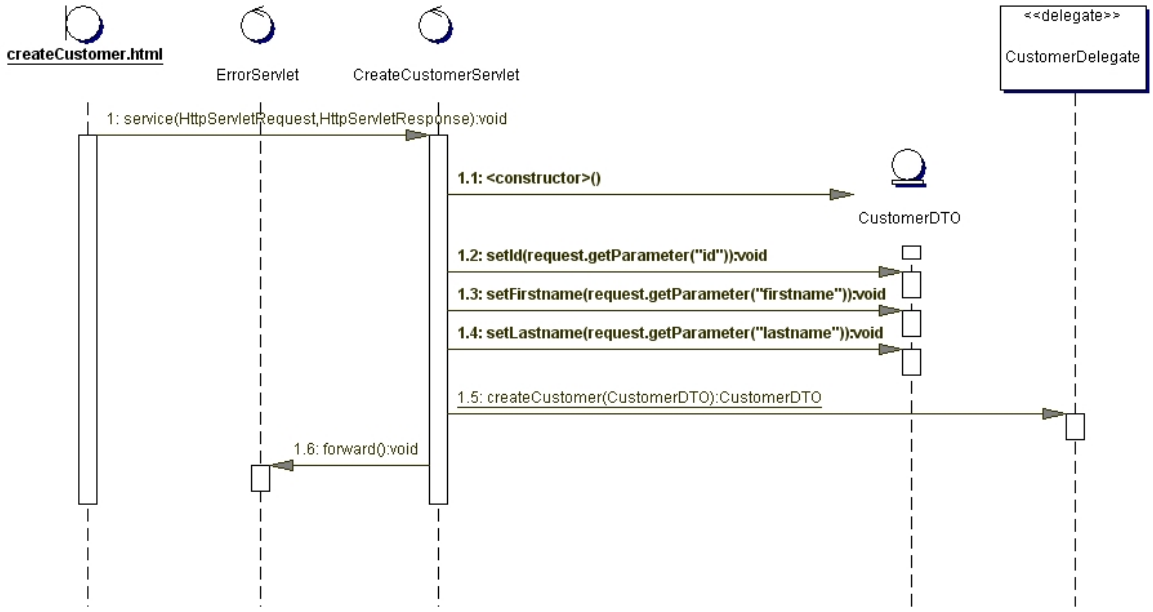


Figure 11 - Sequence d'appel entre la page statique et les deux servlets

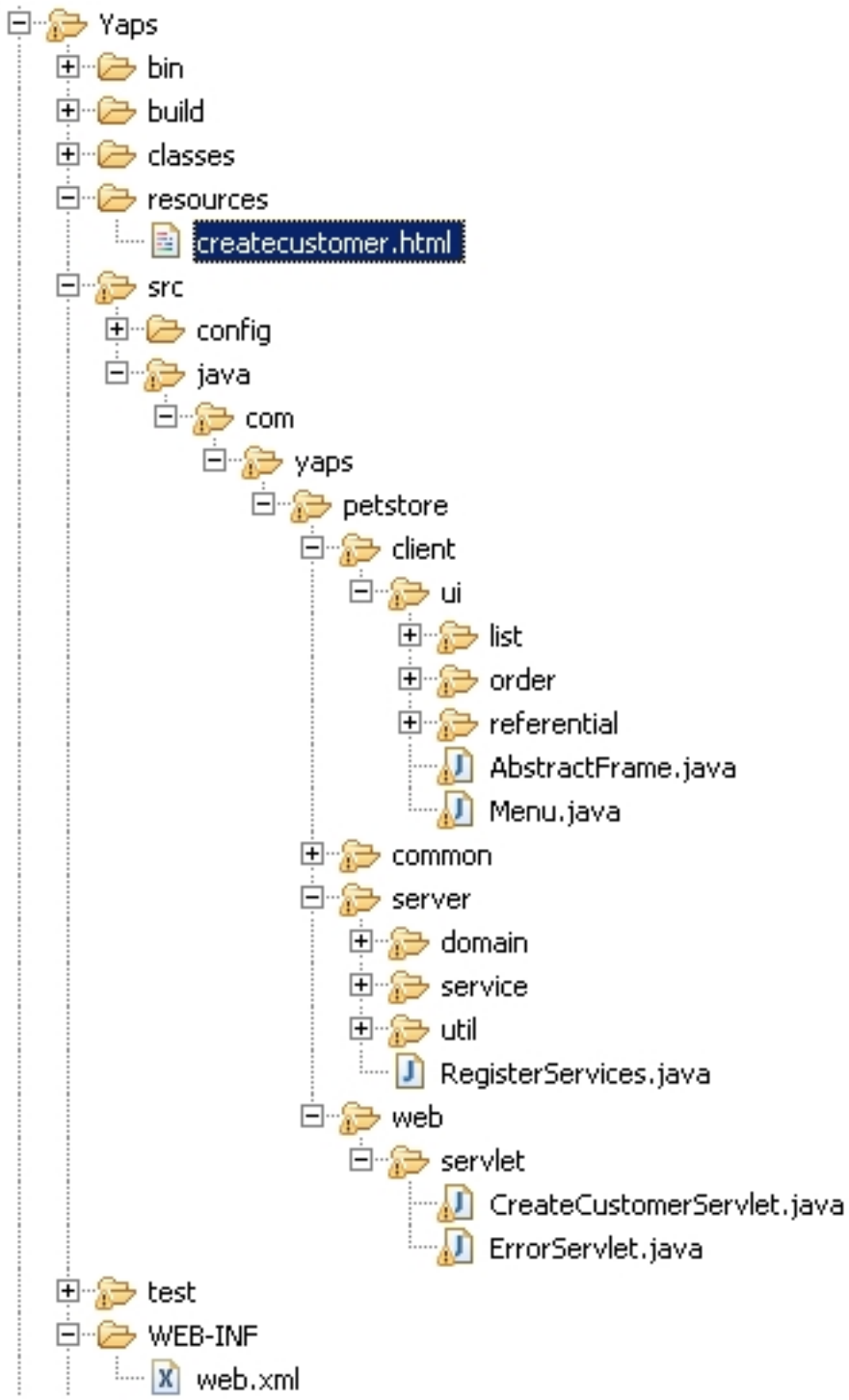
En UML les stéréotypes permettent de créer de nouveaux éléments de modélisation. Ils sont représentés soit par un graphique (comme l'acteur dans les use cases) soit entre guillemets. Tout comme l'acteur, trois autres stéréotypes ont été intégrés à UML

Icône	Stéréotype	Description
	<<boundary>>	représente les objets transitoires qui réalisent les échanges entre le système et les acteurs comme les pages web ou les interfaces graphiques.
	<<control>>	représente un objet fonctionnel créé par le système pour implémenter les mécanismes d'une collaboration comme une servlet.
	<<entity>>	représente les objets métiers du système (objets rendus généralement persistants).

Vue implementation

En plus des paquetages client, common et server, il faut maintenant créer un paquetage accueillant les différents éléments constituant la partie web. Les servlets se trouveront dans le paquetage com.yaps.petstore.web.servlet (alors que les pages statiques HTML seront dans Yaps/resources).

Les interfaces graphiques en mode texte ont été abandonnées, il n'y a plus de raison d'avoir une différence de paquetages (swing et text). L'interface graphique se trouve donc dans com.yaps.petstore.client.ui.



Pour que l'application puisse se compiler et s'exécuter, vous devez rajouter le fichier servlet-api.jar de Tomcat dans votre classpath. Pour que la recette utilisateur puisse se faire, il vous faut aussi rajouter les .jar de HTTPUnit.

Architecture

Le diagramme de composant ci-dessous nous montre comment s'insère le nouveau sous-système servlet ainsi que le navigateur pour les franchisés.

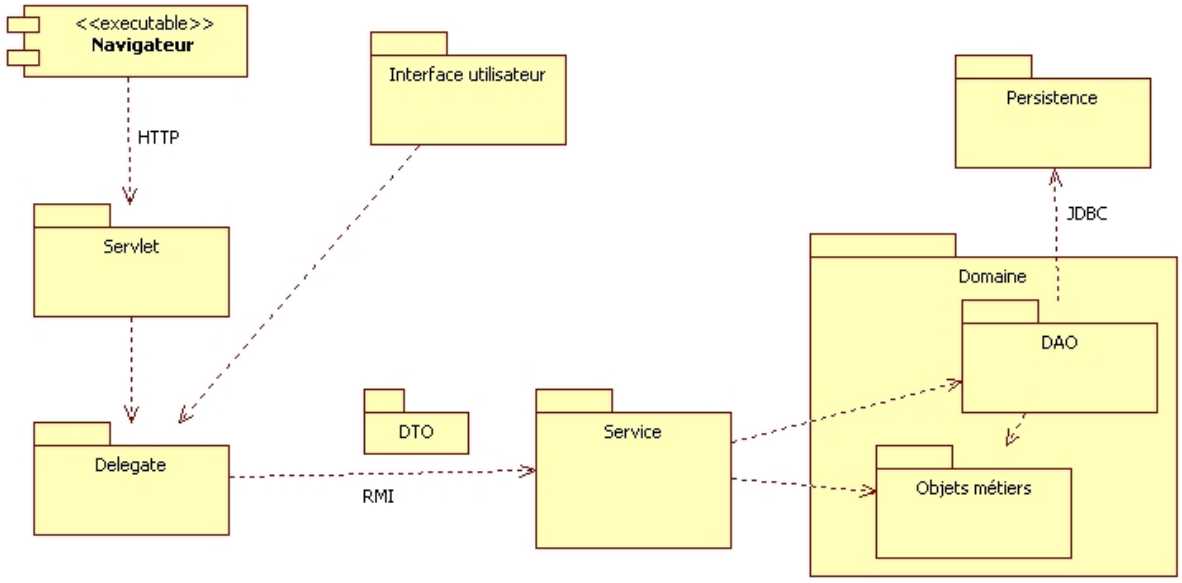


Figure 12 - Diagramme de composant avec le sous-système servlet et le navigateur

Schéma de la base de données

La table t_customer se voit agrémentée de quatre nouvelles colonnes : trois concernant les coordonnées bancaires et une pour stocker l'adresse mail du client. La tache Ant yaps-create-db du fichier build.xml mettra à jour automatiquement cette table.

```
mysql> desc t_customer;
+----------------------+--------------+------+-----+---------+-------+
| Field                | Type         | Null | Key | Default | Extra |
+----------------------+--------------+------+-----+---------+-------+
| id                   | varchar(10)  |      | PRI |         |       |
| firstname            | varchar(50)  |      |     |         |       |
| lastname             | varchar(50)  |      |     |         |       |
| telephone            | varchar(10)  | YES  |     | NULL    |       |
| street1              | varchar(50)  | YES  |     | NULL    |       |
| street2              | varchar(50)  | YES  |     | NULL    |       |
| city                 | varchar(25)  | YES  |     | NULL    |       |
| state                | varchar(25)  | YES  |     | NULL    |       |
| zipcode              | varchar(10)  | YES  |     | NULL    |       |
| country              | varchar(25)  | YES  |     | NULL    |       |
| creditcardnumber     | varchar(25)  | YES  |     | NULL    |       |
| creditcardtype       | varchar(25)  | YES  |     | NULL    |       |
| creditcardexpirydate | varchar(10)  | YES  |     | NULL    |       |
| email                | varchar(255) | YES  |     | NULL    |       |
+----------------------+--------------+------+-----+---------+-------+
```

Vue déploiement

L'application est maintenant déployée sur plusieurs postes. Les employés utilisent l'interface graphique (Petstore Client) qui accède directement au serveur. Les franchisés, eux, utilisent un navigateur qui accède à un serveur web pour la présentation (Petstore Web) et délègue les appels métier au serveur (Petstore Server). Le serveur web Tomcat est hébergé sur un nouvel ordinateur.

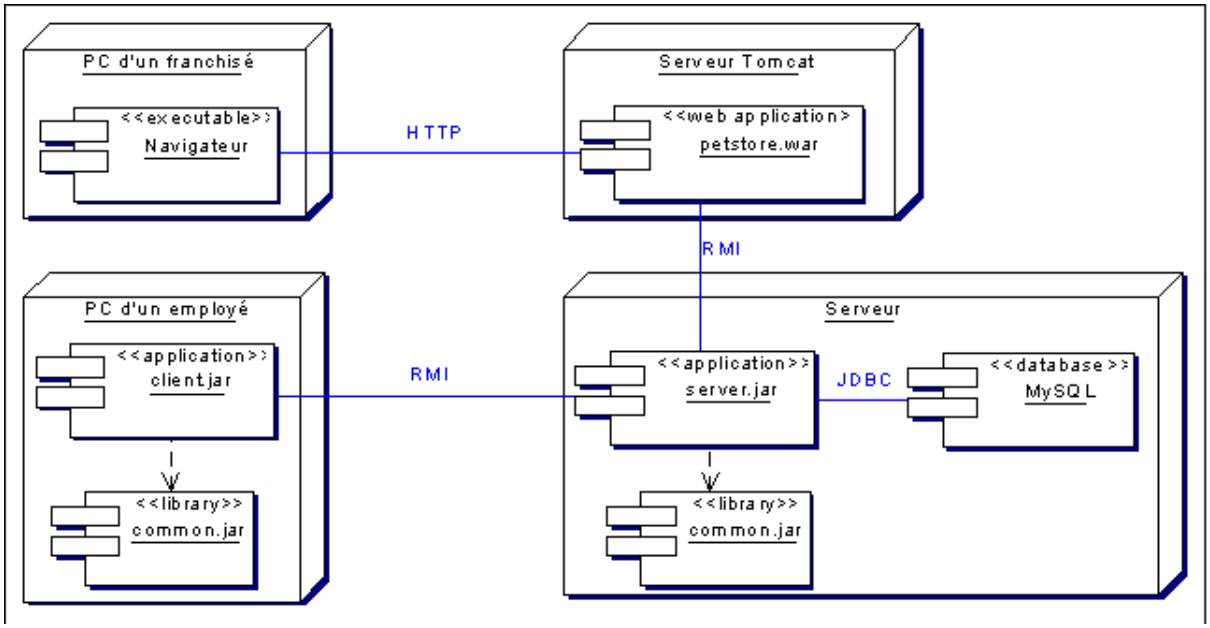


Figure 13 - Déploiement de l'application entre serveur et serveur web

Pour déployer automatiquement l'application web, vous pouvez utiliser la tâche Ant yaps-deploy. Celle-ci copie le fichier petstore.war dans le répertoire webapps de Tomcat. (Pour cela, il faudra définir la variable d'environnement TOMCAT_HOME qui doit préciser le chemin complet du répertoire où Tomcat est installé.)

Le contenu du fichier petstore.war est le suivant : la page HTML est à la racine ainsi que le répertoire WEB-INF contenant le fichier de description web.xml et les classes java compilées utilisées par l'application. Petstore Web ne possède aucune classe métier, celles-ci sont sur le serveur, mais elle y accède via les stubs RMI se trouvant dans le fichier .war. Le fichier web.xml contient la déclaration des deux servlets.

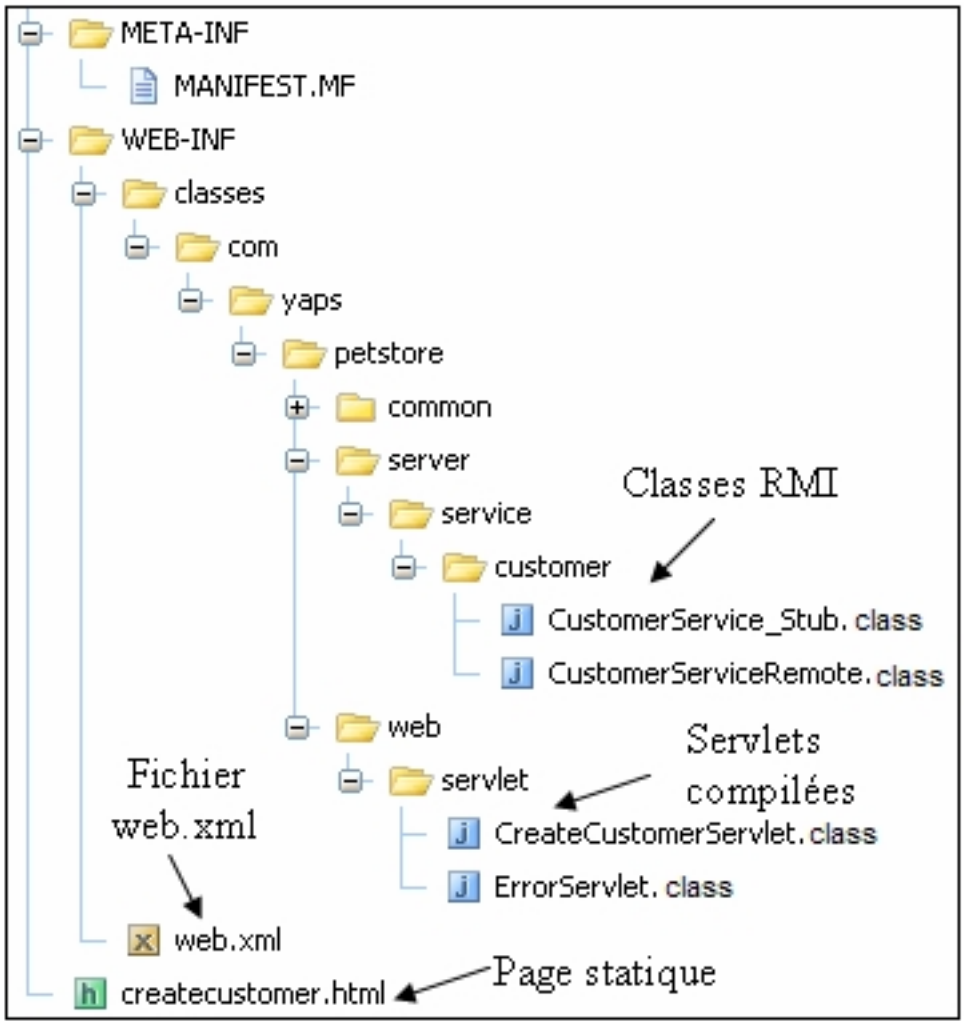


Figure 14 - Contenu du fichier petstore.war

Les fichiers avec une extension .war (Web Application Archive) permettent de regrouper tous les éléments d'une application web que ce soit pour le côté serveur (servlets, classes java...) ou pour le côté client (pages HTML, images, son...). Cette extension du format jar spécialement dédiée aux applications web, a été introduite dans les spécifications de la version 2.2 des servlets. C'est un format indépendant de toute plate-forme et exploitable par tous les conteneurs web qui respectent cette version des spécifications.

Le répertoire WEB-INF et le fichier web.xml qu'il contient doivent obligatoirement être présents dans l'archive. Ce fichier, au format XML, est le descripteur de déploiement qui permet de configurer l'application et les servlets. Le répertoire WEB-INF/classes contient les classes nécessaires à l'application alors que le répertoire WEB-INF/lib contient les librairies externes. Ces deux répertoires sont automatiquement ajoutés par le conteneur au CLASSPATH lors du déploiement de l'application web.

Implémentation

Vous pouvez maintenant développer l'application à partir de la version précédente ou télécharger les classes fournies pour commencer votre développement. Les classes de l'interface graphique utilisée par les employés vous sont presque toutes données. Il vous reste à implémenter l'écran qui gère les produits et les articles du catalogue. La page statique createcustomer.html vous est aussi fournie. À vous de développer la servlet permettant de créer un client (CreateCustomerServlet). Pour vous faciliter le déploiement, le fichier web.xml vous est aussi donné. Ce fichier est placé dans le répertoire Yaps/WEB-INF.

Note : la servlet CreateCustomerServlet peut déléguer l'affichage des erreurs à la servlet ErrorServlet en utilisant un code de cette forme :

```
}
catch (CheckException e) {

String url =
"/error?CheckException: " + e.getMessage();
getServletContext().getRequestDispatcher(url).forward(request, response);
        }
```

Recette utilisateur

La recette utilisateur continue à utiliser les classes de test JUnit pour valider les classes et règles métier de l'application. Par contre, pour la partie web, il nous faut maintenant utiliser les classes de test HTTPUnit.

On pourra ainsi tester la présence des pages statiques ou des servlets dans le serveur web :

```
public class WebTest
extends AbstractTestCase {


public void testCheckHtmlPage() {

try {
       webConversation.getResponse(
"http://localhost:8080/petstore/createcustomer.html");
    }
catch (Exception e) {
        fail();
    }
  }


public void testCheckServlet() {

try {
        webConversation.getResponse(
"http://localhost:8080/petstore/createcustomer");
    }
catch (Exception e) {
        fail();
    }
  }
}
```

On peut aussi effectuer des tests plus poussés. C'est le rôle de la classe CreateCustomerServletTest qui contrôle que la création d'un client par la servlet s'effectue correctement. Elle récupère le formulaire de la page createcustomer.html (1), y insère des données (2) puis simule un clic sur le bouton submit (3). Si cette action entraîne une exception, alors on se trouve sur la page d'erreur ErrorServlet (4).

```
public class CreateCustomerServletTest
extends AbstractTestCase {


private void createCustomer(
final
int id)
throws Exception {
        // Gets the Web Page
        WebResponse customerPage = webConversation.getResponse(
"http://localhost:8080/petstore/createcustomer.html");
        WebForm createCustomerForm = customerPage.getFormWithName(
"customerForm");
        // Sets parameter to the web page
        createCustomerForm.setParameter(
```

```
"id",
"custo" + id);
        createCustomerForm.setParameter(
"firstname",
"firstname" + id);
        createCustomerForm.setParameter(
"lastname",
"lastname" + id);
        createCustomerForm.setParameter(
"email",
"email" + id);
        createCustomerForm.setParameter(
"telephone",
"phone" + id);
        createCustomerForm.setParameter(
"street1",
"street1" + id);
        createCustomerForm.setParameter(
"street2",
"street2" + id);
        createCustomerForm.setParameter(
"city",
"city" + id);
        createCustomerForm.setParameter(
"state",
"state" + id);
        createCustomerForm.setParameter(
"zipcode",
"zip" + id);
        createCustomerForm.setParameter(
"country",
"cnty" + id);
        createCustomerForm.setParameter(
"creditCardExpiryDate",
"10/08");
        createCustomerForm.setParameter(
"creditCardNumber",
"4564 1231 4564 1222");
        createCustomerForm.setParameter(
"creditCardType",
"Visa");
        // Submits the form
        createCustomerForm.submit();
        // If after clicking the submit button an error occurs,
        // the page title is 'YAPS Error'
        WebResponse currentPage = webConversation.getCurrentPage();

if (
"YAPS Error".equals(currentPage.getTitle()))

throw
new Exception(
"An error has occured");
    }
```

Avertissement: Pour que ces classes puissent s'exécuter, le serveur web Tomcat doit tourner avec l'application Petstore Web déployée (ant yaps-deploy). Notre serveur doit aussi être lancé (en exécutant le script startServer.bat du répertoire Yaps/bin dans une fenêtre de commandes dédiée).

Résumé

Dans cette nouvelle version, on utilise une nouvelle interface graphique : un navigateur interprétant le langage HTML. Les design pattern DTO, Business Delegate et Facade permettent à n'importe quel type d'interface graphique d'accéder aux mêmes traitements métiers. En effet, il n'y a pas de traitement particulier propre à telle ou telle interface. Les objets métiers restent identiques et résident sur le serveur.

Pour mettre à disposition des pages HTML et gérer des requêtes HTTP, il faut utiliser un serveur web. En ce qui concerne la possibilité d'intercepter ces requêtes et d'effectuer un traitement (dans notre cas un appel au Business Delegate) les servlets sont appropriées.

Références

Java Servlet Technology http://java.sun.com/javaee/6/docs/tutorial/doc/bnafd.html

Javadoc (EE6) http://java.sun.com/javaee/6/docs/api/index.html

Java Servlet Programming, 2nd Edition Jason Hunter. O'Reilly. 2001.

Hypertext Transfer Protocol http://www.w3.org/Protocols/

HTML Home Page http://www.w3.org/MarkUp/

HTML Primer http://www.htmlprimer.com/

Tomcat http://tomcat.apache.org/

Professional Apache Tomcat 5 (Programmer to Programmer) Vivek Chopra, Amit Bakore, Jon Eaves, Ben Galbraith, Sing Li, Chanoch Wiggers. Wrox. 2004

HTTPUnit http://httpunit.sourceforge.net/

Extract Class refactoring http://www.refactoring.com/catalog/extractClass.html

Hide Delegate refactoring http://www.refactoring.com/catalog/hideDelegate.html

Extract Class refactoring http://www.refactoring.com/catalog/selfEncapsulateField.html

Singleton Pattern http://en.wikipedia.org/wiki/Singleton_pattern