TP 4 : Interface graphique

Pascal GRAFFION
2013/11/07 15:43

Table des matières

TP 4 : Interface graphique

Enormément de programmes Java destinés aux utilisateurs, comportent des éléments graphiques de telle sorte que l'application soit plus conviviale et ergonomique au travers de son Interface Homme-Machine (IHM). La plate-forme Java dispose de différents outils, tels que AWT et Swing, permettant de construire des interfaces graphiques sophistiquées.

AWT est antérieur à Swing. AWT a été développée pour la première sortie de Java version 1.0 du JDK alors que Swing n'est apparu qu'à la version 1.2 (soit Java 2). Il en résulte donc des différences fondamentales de conception entre les deux librairies.

Un composant AWT lors de sa création est associé à une fenêtre distincte gérée par le système d'exploitation sous-jacent. Et c'est le système d'exploitation qui est responsable de son apparence. Cette « manière » de faire, bien qu'elle ait fait ses preuves et qu'elle ait permis au langage Java de s'imposer, est très lourde (perte de performance, et consommation excessive de mémoire). C'est pour cette raison que l'on qualifie les composants AWT de heavyweight (littéralement, poids lourds)

Par opposition, les composants Swing sont simplement dessinés à l'intérieur de leur conteneur comme s'il s'agissait d'une image. Le système d'exploitation n'est pas sollicité mais uniquement la machine virtuelle. C'est pourquoi ils sont qualifiés de lightweight (composants allégés).

L'avantage des composants Swing par rapport à AWT, c'est qu'ils n'utilisent plus aucun code natif, les possibilités offertes deviennent alors plus larges. On peut, par exemple, faire en sorte que les boutons présentent une image à la place d'un texte. On peut avoir des boutons ronds, on peut créer des bords variés pour les composants...

Les objets de la classe java.awt.Container de AWT sont conçus pour pouvoir contenir des composants graphiques (java.awt.Component). Tous les composants Swing sont dérivés de la classe JComponent. Cette dernière est une classe dérivée de la classe java.awt.Component. Autrement dit, n'importe quel composant Swing peut contenir un ou plusieurs autres composants AWT ou Swing. Cette approche permet de définir des composants beaucoup plus riches.

Hello PetStore !

Le code qui suit affichera un panneau comportant une zone de texte (1) et deux boutons (2). Lorsque vous cliquez sur ces boutons (3) (4), une chaîne de caractère s'affiche dans la zone de texte.

```
import javax.swing.*;

import java.awt.*;

import java.awt.event.ActionListener;

import java.awt.event.ActionEvent;


public class HelloPetstore
extends JFrame {

    // Une zone de texte et deux boutons

private
final JTextField textHelloPetstore =
new JTextField(); (1)

private
final JButton buttonHello =
new JButton(
"Hello");

private
final JButton buttonPetstore =
new JButton(
"Petstore"); (2)
```

```
public HelloPetstore() {

        // Une action est déclenchée lorsqu'on clique sur le bouton Hello
        buttonHello.addActionListener(
new ActionListener() {

public void actionPerformed(
final ActionEvent evt) {
                textHelloPetstore.setText(
"Hello"); (3)
            }
        });

        // Une action est déclenchée lorsqu'on clique sur le bouton Petstore
        buttonPetstore.addActionListener(
new ActionListener() {

public void actionPerformed(
final ActionEvent evt) {
                textHelloPetstore.setText(
"PetStore !!!"); (4)
            }
        });

        // Disposition des éléments graphiques
        getContentPane().setLayout(
new GridLayout(3, 1));
        getContentPane().add(textHelloPetstore);
        getContentPane().add(buttonHello);
        getContentPane().add(buttonPetstore);
        pack();
    }

    // Point d'entrée de l'application

public
static void main(
String[] args) {

new HelloPetstore().show();
    }
}
```

Expression des besoins

Les bénéfices de YAPS ne cessent de s'accroître. Les ventes sont en hausse et la société se doit d'embaucher de nouveaux employés pour faire face à cette envolée du marché. YAPS veut maintenant informatiser la saisie d'un bulletin de commande. Ce dernier est créé lorsqu'un client achète au moins un article, c'est-à-dire un animal de compagnie. Il fait aussi office de bon de livraison puisqu'on y indique l'adresse où les animaux doivent être livrés. Ce bulletin de commande possède un numéro unique, une date et fait référence à un client, celui qui est à l'origine de la commande.

John et Bill continuent à s'occuper respectivement de la gestion des clients et du catalogue de la société alors que les nouveaux employés se chargeront de passer les commandes. Par contre, ils ne trouvent pas leurs interfaces très conviviales, et veulent utiliser des interfaces graphiques un peu plus sophistiquées. Bien que ces employés ne puissent pas créer ou modifier des clients ou des articles du catalogue, ils veulent avoir la possibilité d'en consulter la liste. Ce système de gestion des commandes se nomme PetStore Order.

Vue Utilisateur

Diagramme de cas d'utilisation

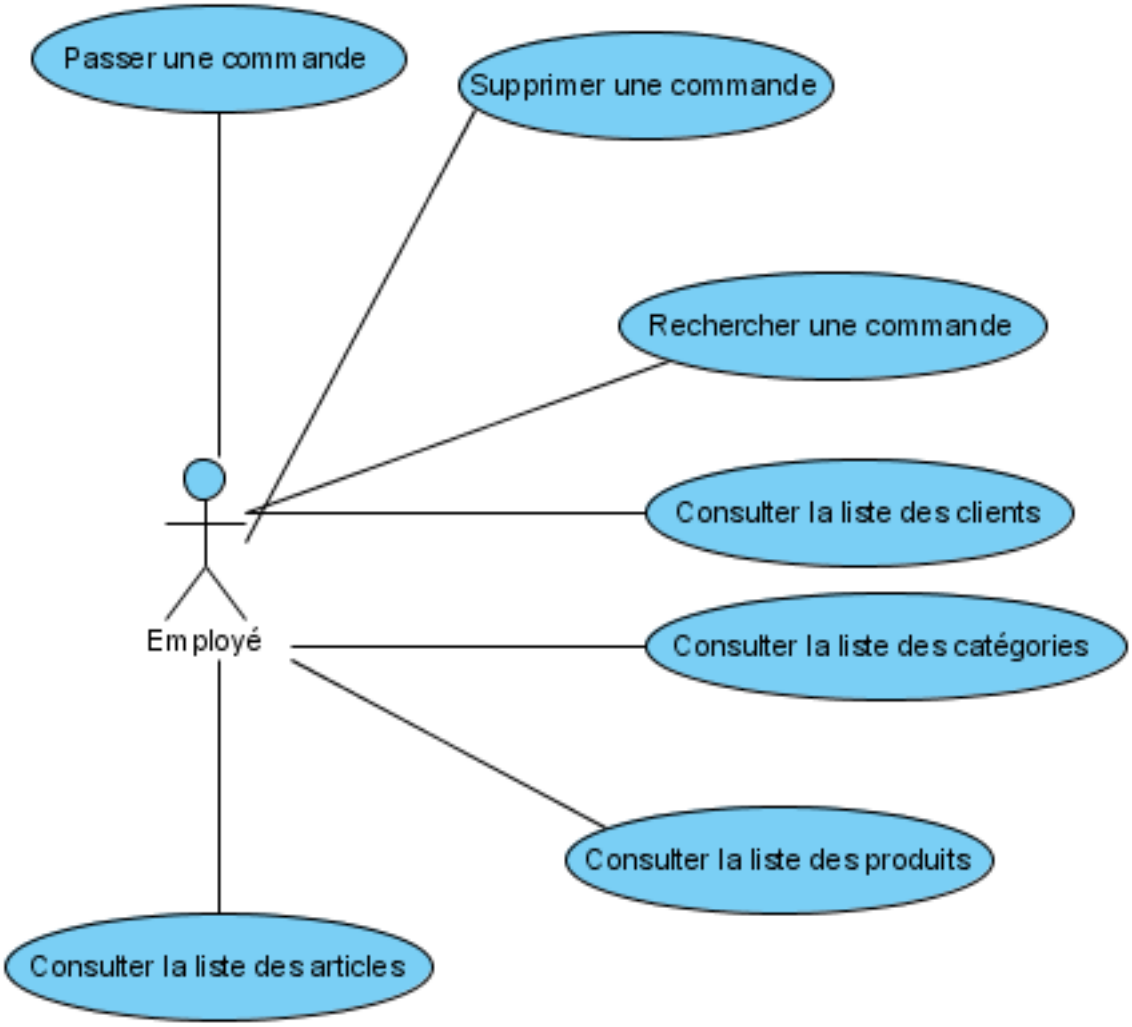


Figure 1 - Diagramme de cas d'utilisation des commandes

Dans les précédents cas d'utilisation, les acteurs étaient clairement définis par leur prénom (Bill et John). YAPS a maintenant embauché plusieurs nouveaux employés, il serait inapproprié de nommer tous les acteurs du système par leur prénom. Dans le cas d'utilisation ci-dessus, l'acteur est nommé « employé » ce qui signifie que n'importe quel employé peut créer un bon de commande, le supprimer ou le rechercher.

Les quatre cas d'utilisation de consultation (clients, catégories, produits et articles) n'ont pas besoin d'être détaillées. Ils sont identiques à ceux vus précédemment pour les applications Petstore Customer et Petstore Catalog.

Cas d'utilisation « Passer une commande »

Nom	**Passer une commande.**
Résumé	Permet à un employé de la société YAPS de créer une commande pour un client.
Acteurs	Employé.
Pré-conditions	Une commande est créée pour un client qui achète de un à cinq articles. Ce client et ces articles doivent exister dans le système.
Description	Lorsqu'un employé est contacté au téléphone par un client pour passer une commande (Order), ce client doit être connu du système ((1)). L'employé lui demande alors son identifiant. Cet identifiant doit apparaître dans le bon de commande. Le client a la possibilité d'acheter jusqu'à cinq articles ((2)), représentant jusqu'à cinq lignes de commande (Order Line). L'employé saisit donc les identifiants des articles ((3)) ainsi que la quantité. Le système est alors capable de ramener les informations de l'article (nom et prix unitaire), de calculer le sous total (prix unitaire de l'article

	multiplié par la quantité) ainsi que le total de la commande (somme des articles achetés par le client). Le client peut acheter directement par téléphone en donnant ses coordonnées bancaires ou envoyer un chèque. Lors de la création de la commande, un numéro séquentiel est attribué par le système. Ce numéro unique permettra aux employés de retrouver la commande. Une fois le paiement perçu, les animaux domestiques sont envoyés à l'adresse de la commande. Il est à noter qu'un client peut acheter des articles pour une autre personne à une autre adresse que la sienne. Il est aussi important de préciser que pour la comptabilité, toutes les commandes sont archivées sur plusieurs années. Le prix d'un article peut varier dans le temps, mais le prix qui a permis d'effectuer la commande doit être préservé. Les données d'une commande sont les suivantes : - Order Id : identifiant unique d'une commande. Cet identifiant est construit automatiquement par le système. - Order Date : date à laquelle la commande a été créée. Cette date est renseignée automatiquement par le système. - Customer Id : Une référence vers le client qui a passé la commande - First Name : prénom de la personne à qui livrer les articles (par défaut c'est le prénom du client) - Last Name : nom de famille de la personne à qui livrer les articles (par défaut c'est le nom du client) - Street 1 et Street 2 : ces deux zones permettent de saisir l'adresse de livraison (par défaut, c'est l'adresse du client). - City : ville de livraison (par défaut, ville de résidence du client) - State : état de livraison (uniquement pour les clients américains) - Zipcode : code postal de l'adresse de livraison (par défaut, code postal du client) - Country : pays de livraison (par défaut, pays de résidence du client) - 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 '/'. Une à cinq lignes de commande. Chaque ligne de commande fait référence à un article, à une quantité et au prix de l'article (le prix de l'article peut évoluer dans le temps, il faut donc sauvegarder le prix)
Exceptions	((1)) Si l'identifiant du client n'est pas présent dans le système une exception ObjectNotFoundException est levée. ((2)) Si le nombre d'articles est inférieur à un, une exception CheckException est levée. ((3)) Si l'identifiant de l'article n'est pas présent dans le système une exception ObjectNotFoundException est levée. ((GLOBAL)) Si une erreur système se produit durant la création d'un produit, l'exception CreateException doit être levée.
Post-conditions	Une commande est créée.

Diagramme d'activité

Ci-dessous le diagramme d'activité donne une vision plus concise du cas d'utilisation.

Le diagramme d'activité est utilisé pour représenter des aspects dynamiques d'un système à un niveau assez général.

Il est composé d'un neud initial (représenté par un point noir), d'activités liées entre elles par des événements, puis se termine par un noeud final (un rond noir entouré).

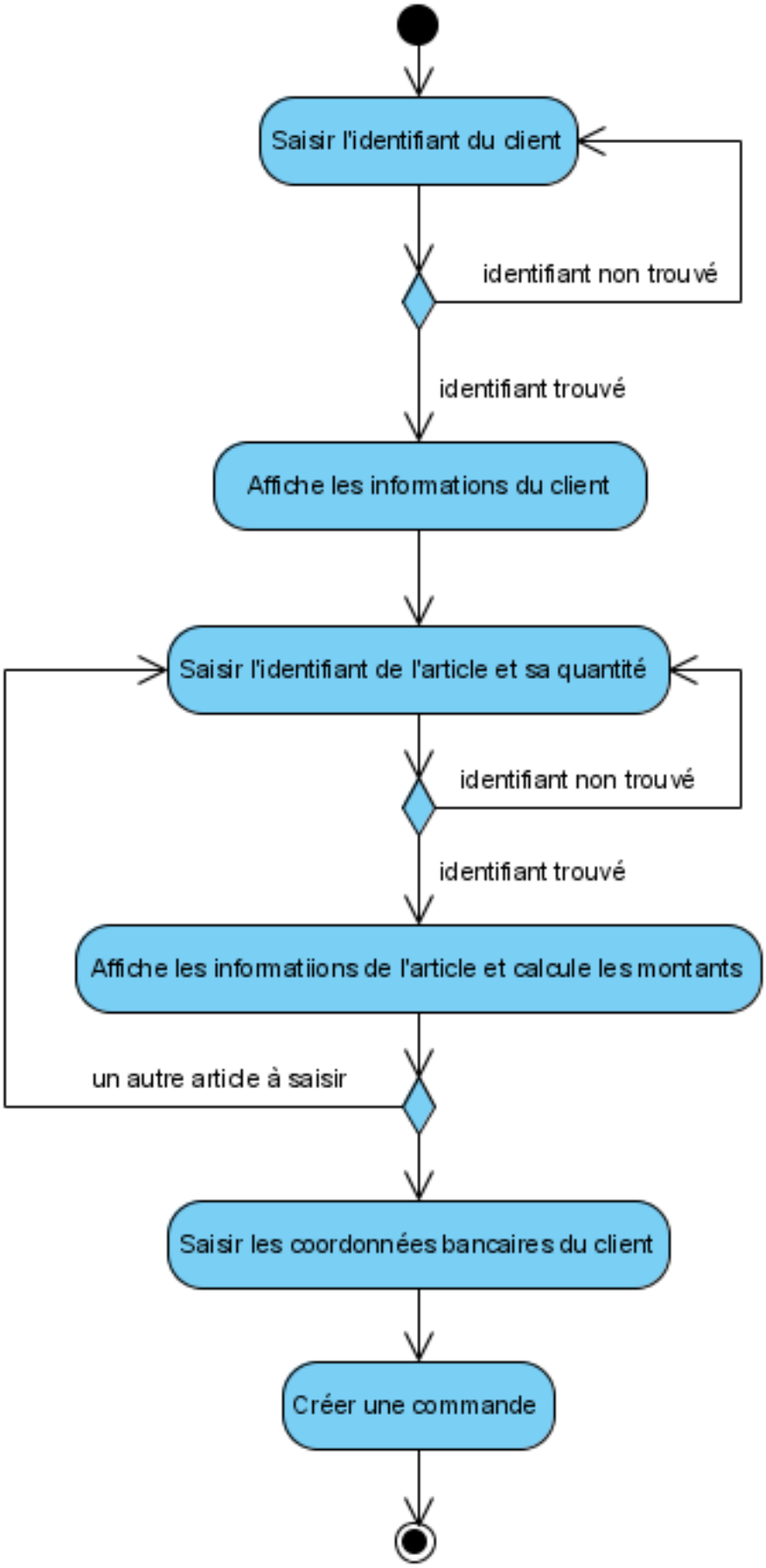

Saisir l'identifiant du client
identifiant non trouvé
identifiant trouvé
Affiche les informations du client
Saisir l'identifiant de l'article et sa quantité
identifiant non trouvé
identifiant trouvé
Affiche les informatiions de l'article et calcule les montants
un autre article à saisir
Saisir les coordonnées bancaires du client
Créer une commande

Figure 2 - Diagramme d'activité pour passer une commande

Cas d'utilisation « Rechercher une commande »

Nom	**Rechercher une commande par son identifiant.**
Résumé	Permet à un employé de retrouver une commande.
Acteurs	Employé.
Pré-conditions	La commande doit être existante dans le système ((1)).
Description	A partir d'un numéro de commande (Order Id), le système affiche les détails suivants : identifiant du client qui a demandé la commande, coordonnée pour la livraison, coordonnées bancaires, liste les lignes de commande (Order Line) composant la commande et le total de celle-ci.
Exceptions	((1)) Si l'identifiant n'est pas présent dans le système une exception ObjectFoundException est levée. ((GLOBAL)) Si une erreur système se produit, l'exception FinderException doit être levée.
Post-conditions	Les détails de la commande sont affichés.

Cas d'utilisation « Supprimer une commande »

Nom	**Supprimer une commande.**
Résumé	Permet à un employé de supprimer une commande du système.
Acteurs	Employé.
Pré-conditions	La commande doit être existante dans le système ((1)).
Description	A partir d'un numéro de commande (Order Id) le système en affiche les détails et propose à l'employé de la supprimer. Si l'employé accepte alors la commande est supprimé du système. Le système doit aussi s'assurer que les lignes de commande doivent aussi être supprimées.
Exceptions	((1)) Si l'identifiant n'est pas présent dans le système une exception ObjectNotFoundException est levée. ((GLOBAL)) Si une erreur système se produit, l'exception RemoveException doit être levée.
Post-conditions	La commande et ses lignes de commande sont supprimées.

Ecrans

Les menus MenuCatalog et MenuCustomer utilisés par John et Bill ne changent pas dans cette version du PetStore. Par contre, une nouvelle interface utilisateur est créée pour que les employés puissent saisir des commandes. Cette interface se compose d'un menu principal permettant d'accéder à différents sous-menus.

Figure 3 - Menu principal

Menu	Sous-Menu
File	Exit : permet à l'employé de fermer l'application
Order	Create Order : affiche l'écran de création d'une commande
	Manage Order : affiche l'écran permettant à l'employé de retrouver et de supprimer une commande
Lists	List customers : affiche la totalité des clients de l'application
	List categories : affiche la totalité des catégories du catalogue
	List products : affiche la totalité des produits du catalogue
	List items : affiche la totalité des articles du catalogue
Look&Feel	Metal : change l'aspect de l'application en Metal
	Motif : change l'aspect de l'application en Motif
	Windows : change l'aspect de l'application en Windows

Les écrans de liste permettent aux employés d'avoir la liste des clients et des éléments du catalogue à portée de main. Ils ne permettent pas la modification (cette tâche étant exclusivement réservée à Bill et John) mais simplement la consultation. Ci-dessous un exemple des écrans listant les catégories et les produits.

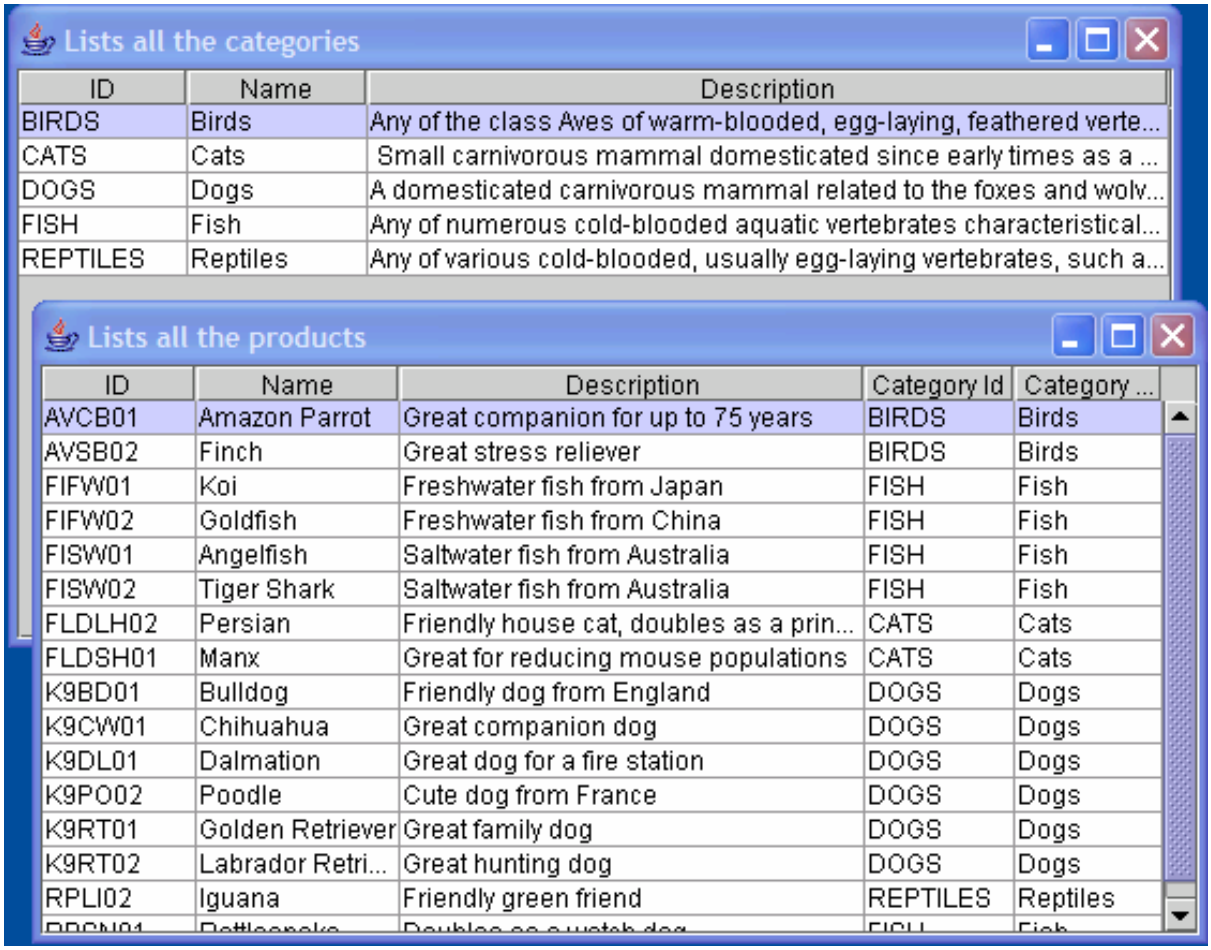


Figure 4 - Ecrans listant les catégories et les produits

L'écran de création d'une commande est plus compliqué et demande à l'employé d'accomplir plusieurs étapes.

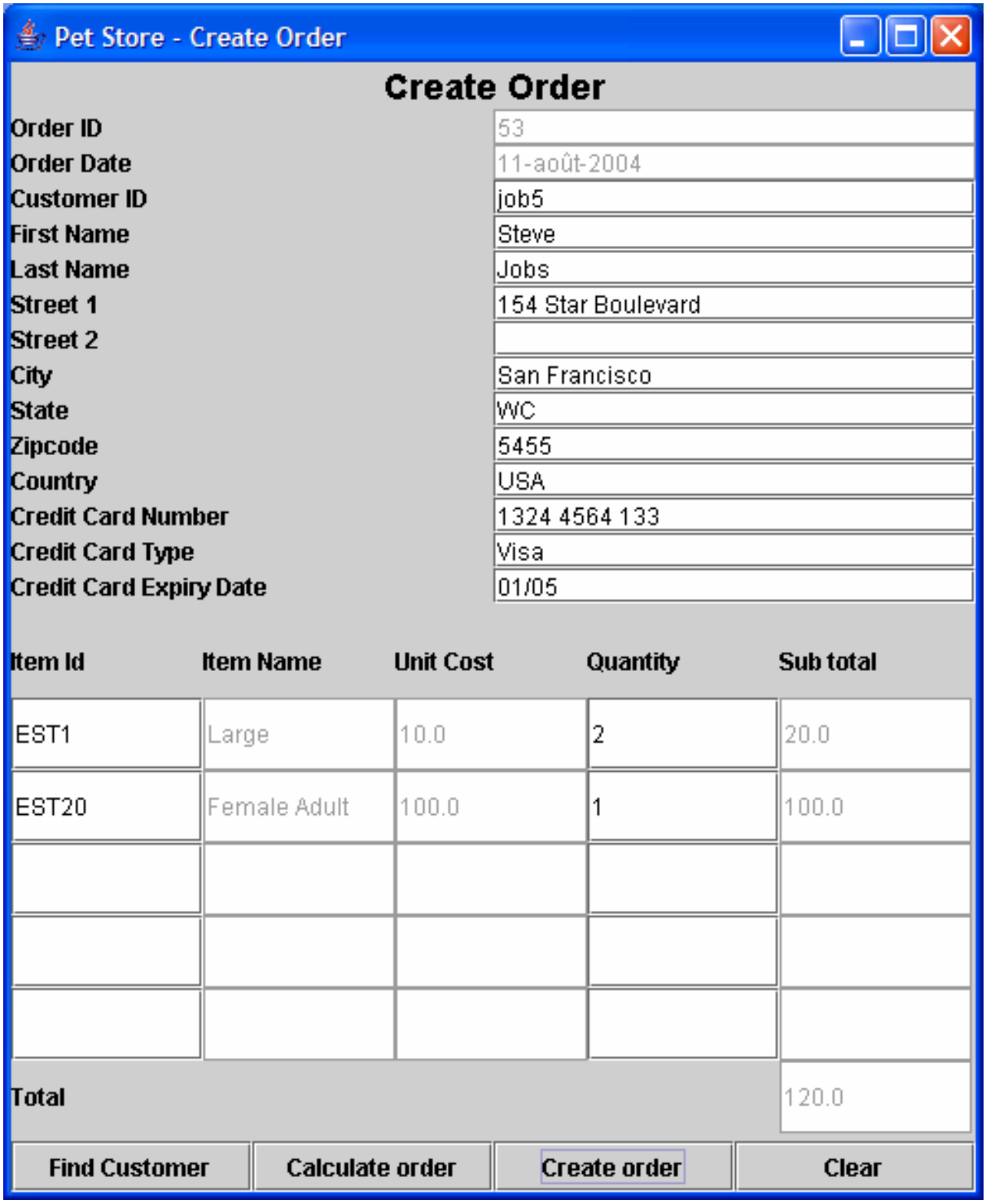


Figure 5 - Ecran de saisie d'une commande

Après avoir saisi l'identifiant du client (1), l'employé clique sur le bouton Find Customer (2). Cette action permet au système d'afficher les coordonnées du client (3). Ensuite l'employé saisit les identifiants des articles (4) que le client désire acheter ainsi que la quantité. En cliquant sur Calculate Order (5) le système ramène les informations relatives à l'article (6) et calcule les différents totaux (7). Et c'est seulement lors du clic sur Create Order (8) que la commande est enregistrée dans le système. Ce n'est alors qu'à ce moment-là que son numéro et sa date de création s'affichent (9). Le bouton Clear vide les zones de saisie de l'écran.

Le deuxième écran, Manage Order, permet aux employés de retrouver une commande par rapport à son identifiant et de la supprimer le cas échéant.

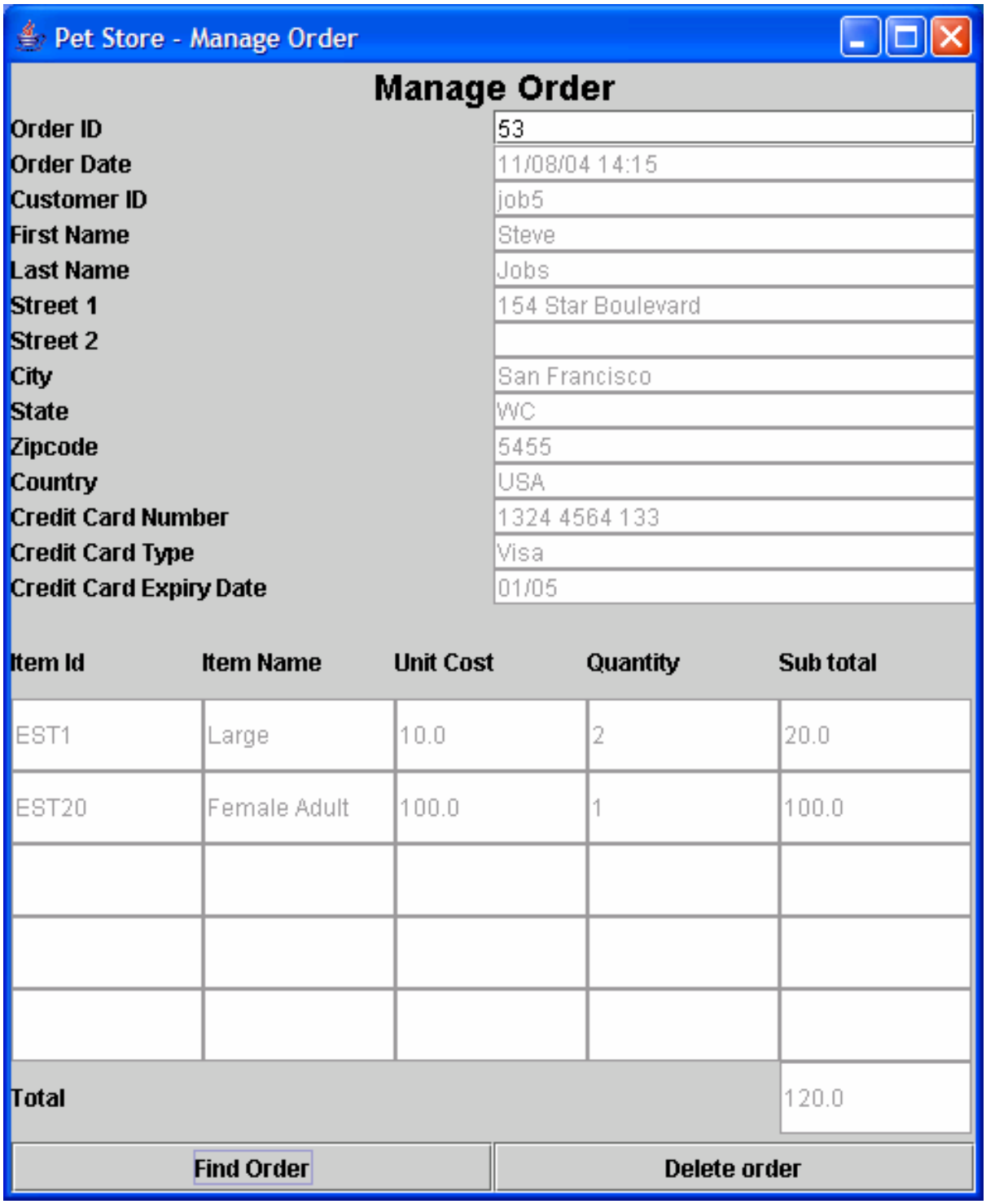


Figure 6 - Ecran de recherche et de suppression d'une commande

Pour retrouver une commande, l'employé saisit son numéro (1) puis clique sur le bouton Find Order (2). Il peut ensuite la supprimer en cliquant sur Delete Order (3).

Analyse et conception

Vue logique

Le diagramme de classes ci-dessous vous montre la classe commande (Order) et ses différentes relations. Une commande est passée par un client et contient plusieurs lignes de commande. Chaque ligne se rapporte à un article.

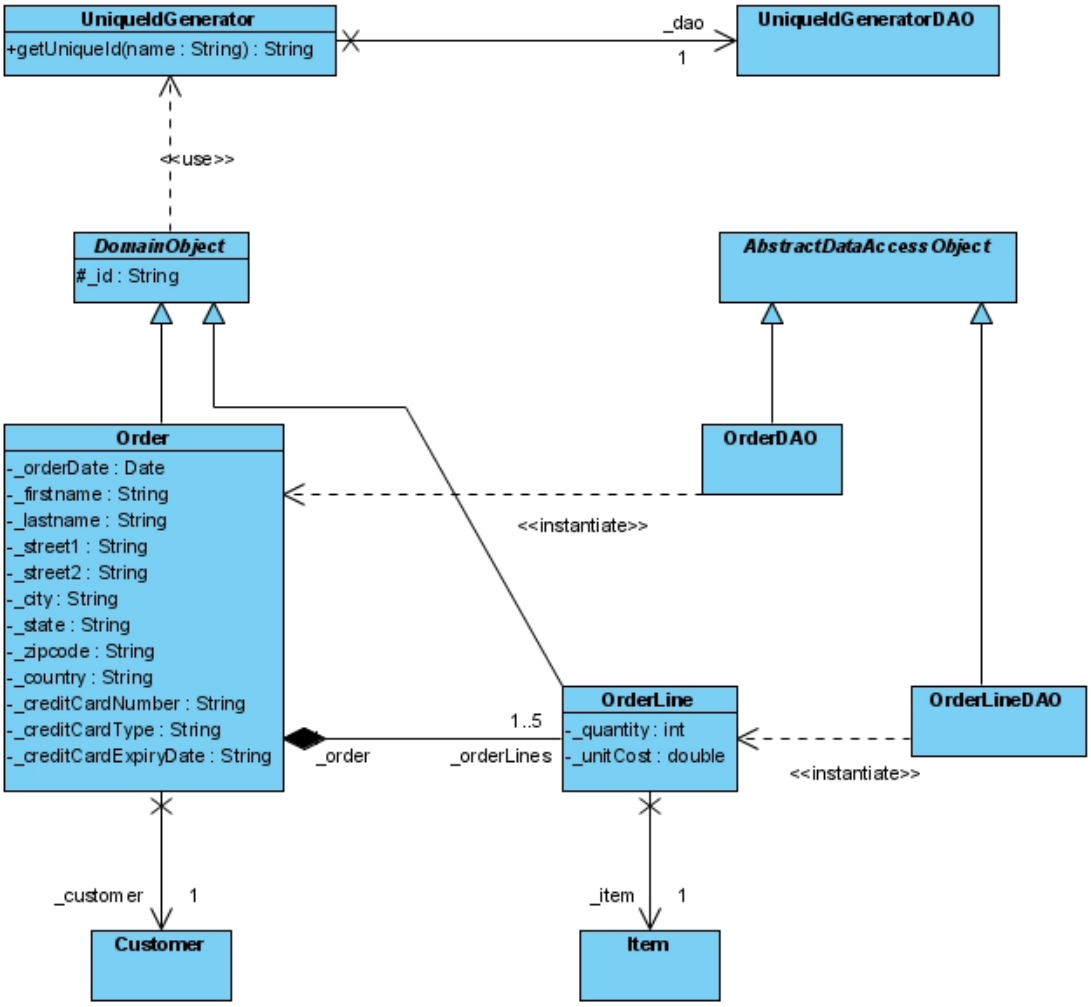


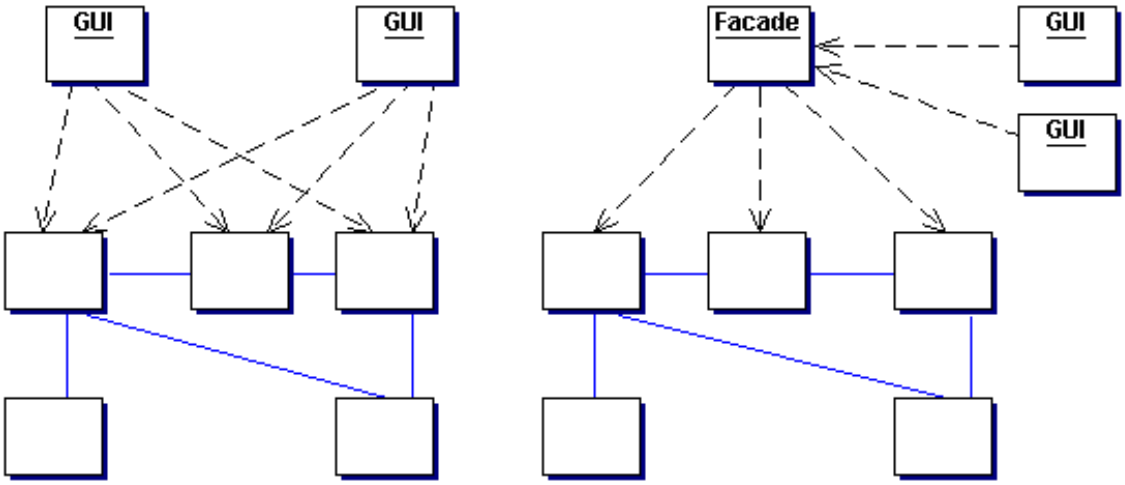
Figure 7 - Diagramme de classe d'une commande

Pour les clients, les catégories, les produits et les articles, Bill et John se chargent de créer un identifiant unique. Pour ce qui est des commandes ou des lignes de commande, ce numéro doit être généré automatiquement par le système. Pour cela, la méthode getUniqueId() de la classe AbstractDataAccessObject utilisera la classe UniqueIdGenerator. Celle-ci s'appuiera sur une table en base pour persister le dernier identifiant. Une fois la méthode getUniqueId() appelée, cette classe se chargera de récupérer l'identifiant, de l'incrémenter de un et de le re-stocker en base. Elle déléguera la persistance à la classe UniqueIdGeneratorDAO. La méthode getUniqueId() prend une chaîne de caractères en paramètre. Par exemple getUniqueId(« Order ») donnera un identifiant unique pour les commandes, alors que getUniqueId(« OrderLine ») sera utilisé pour les lignes de commande.

Vue Processus

Jusqu'à présent les interfaces utilisateurs traitaient des choses bien différentes : MenuCustomer gère uniquement les clients et, MenuCatalog, le catalogue de la société. Maintenant l'application Petstore Order permet de gérer les commandes mais aussi d'afficher la liste des clients et des éléments du catalogue. On se retrouve donc à dupliquer du code métier dans plusieurs interfaces utilisateurs. Pour palier ce problème, on peut utiliser le design pattern Facade.

Le design pattern Facade permet de fournir un point d'entrée simple à un système complexe. Il introduit une interface pour découpler les relations entre deux systèmes : dans notre cas l'interface graphique et le traitement métier.



Dans le diagramme de gauche, les interfaces graphiques dialoguent directement avec les objets métiers. Si on utilise cette approche, l'appel au code java permettant d'obtenir la liste des clients, par exemple, serait dupliqué dans les deux interfaces graphiques. Grâce à la façade (diagramme de droite), les deux interfaces n'ont qu'à appeler une seule classe (la façade) qui elle, appellera les objets métiers.
Pour illustrer ce design pattern, reprenons le cas de la création d'un produit vu précédemment. Les diagrammes suivants vous donnent les séquences d'appels en utilisant une façade (figure 8), ici représentée par la classe CatalogService, et sans façade (figure 9). Pour simplifier la lecture nous n'avons pas toujours ajouté les classes DAO. Après avoir instancié les objets Product et Category avec leur valeur, un unique appel à la façade se chargera de créer un produit, c'est-à-dire rechercher la catégorie, la lier au produit et créer ce dernier.

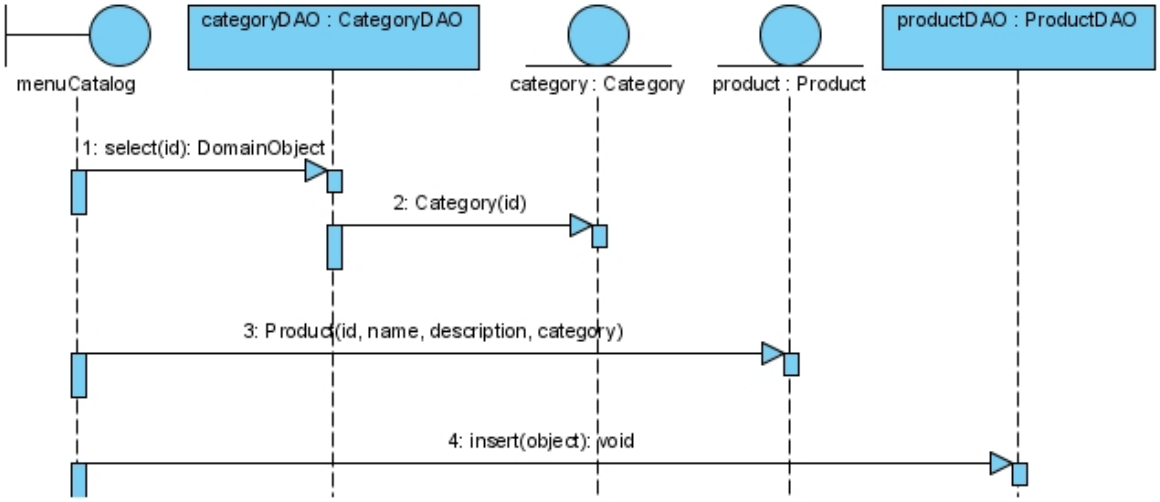


Figure 8 -- Diagramme de séquence pour créer un produit sans façade

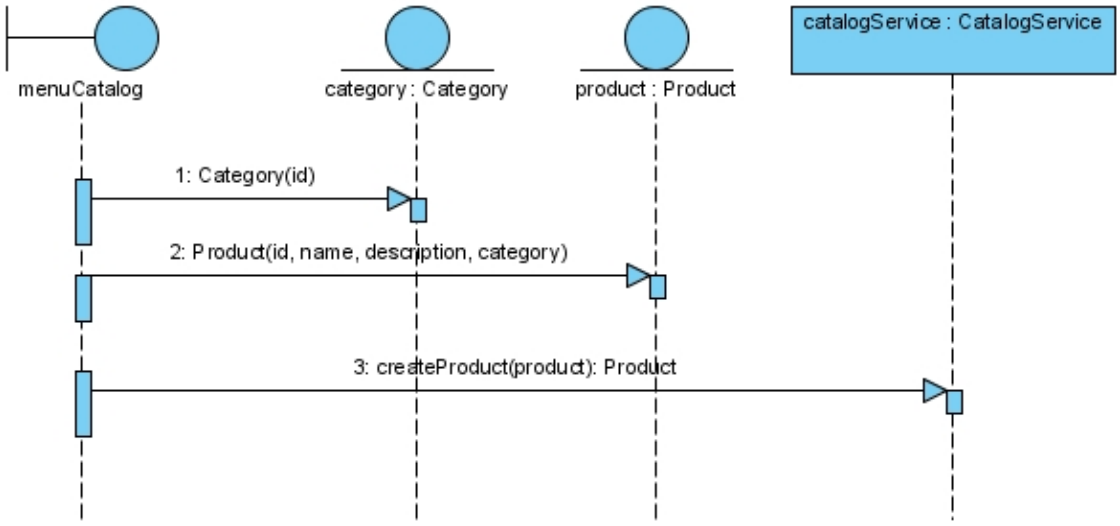


Figure 9 - Diagramme de séquence pour créer un produit avec façade

Grâce à ce design pattern, le code gérant l'appel aux différents objets est déporté à un seul endroit et peut être réutilisé par différentes interfaces utilisateurs. On peut donc créer trois façades : CatalogService, CustomerService et OrderService. Ci-dessous les signatures de ces trois classes :

```
public
final class CatalogService {
    Category createCategory(Category category)
throws CreateException, CheckException;
    Category findCategory(
String categoryId)
throws FinderException, CheckException;
    void deleteCategory(
String categoryId)
throws RemoveException, CheckException;
    void updateCategory(Category category)
throws UpdateException, CheckException;
    Collection findCategories()
throws FinderException;
    Product createProduct(Product product)
throws CreateException, CheckException;
    Product findProduct(
String productId)
throws FinderException, CheckException;
    void deleteProduct(
String productId)
throws RemoveException, CheckException;
    void updateProduct(Product product)
throws UpdateException, CheckException;
    Collection findProducts()
throws FinderException;
    Item createItem(Item item)
throws CreateException, CheckException;
    Item findItem(
String itemId)
throws FinderException, CheckException;
    void deleteItem(
String itemId)
throws RemoveException, CheckException;
    void updateItem(Item item)
throws UpdateException, CheckException;
    Collection findItems()
throws FinderException;
}

public
final class CustomerService {
    Customer createCustomer(Customer customer)
```

```
throws CreateException, CheckException;
    Customer findCustomer(
String customerId)
throws FinderException, CheckException;
    void deleteCustomer(
String customerId)
throws RemoveException, CheckException;
    void updateCustomer(Customer customer)
throws UpdateException, CheckException;
    Collection findCustomers()
throws FinderException;
}

public
final class OrderService {
    Order createOrder(Order order)
throws CreateException, CheckException;
    Order findOrder(
String orderId)
throws FinderException, CheckException;
    void deleteOrder(
String orderId)
throws RemoveException, CheckException;
}
```

L'autre aspect intéressant de ce design pattern est le fait de « masquer » des fonctionnalités aux classes clientes. Par exemple, la façade OrderService ne manipule que des Orders alors que les classes OrderDAO et OrderLineDAO (que la façade OrderService utilise) peuvent insérer en base, modifier ou supprimer des instances d'Order et OrderLine. En conséquence, l'interface utilisateur de gestions des commandes ne permet pas la modification d'une ligne de commande.

Vue implémentation

Les classes façades se trouvent dans un paquetage service et les interfaces utilisateurs sont maintenant différenciées entre mode graphique et mode texte dans des paquetages différents.

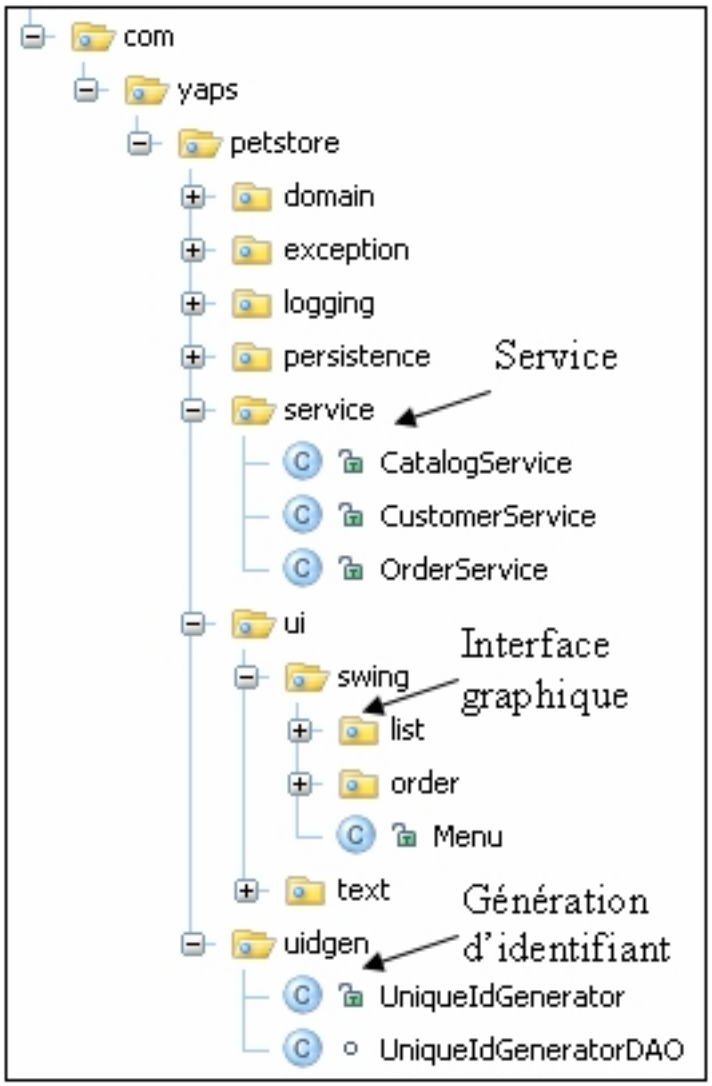


Figure 10 - Paquetage de l'application

Architecture

Le diagramme de composants ci-dessous nous montre comment s'insère le sous-système service dans l'architecture.

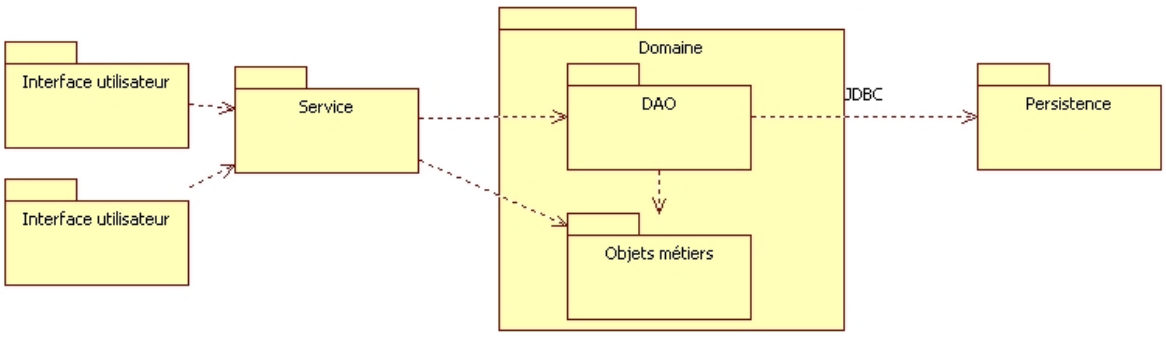


Figure 11 - Diagramme de composants avec le sous-système service

Schéma de la base de données

Cette nouvelle version de l'application PetStore a besoin de 3 nouvelles tables pour stocker les informations des commandes, des lignes de commande et les identifiants uniques (t_order, t_order_line et t_counter).

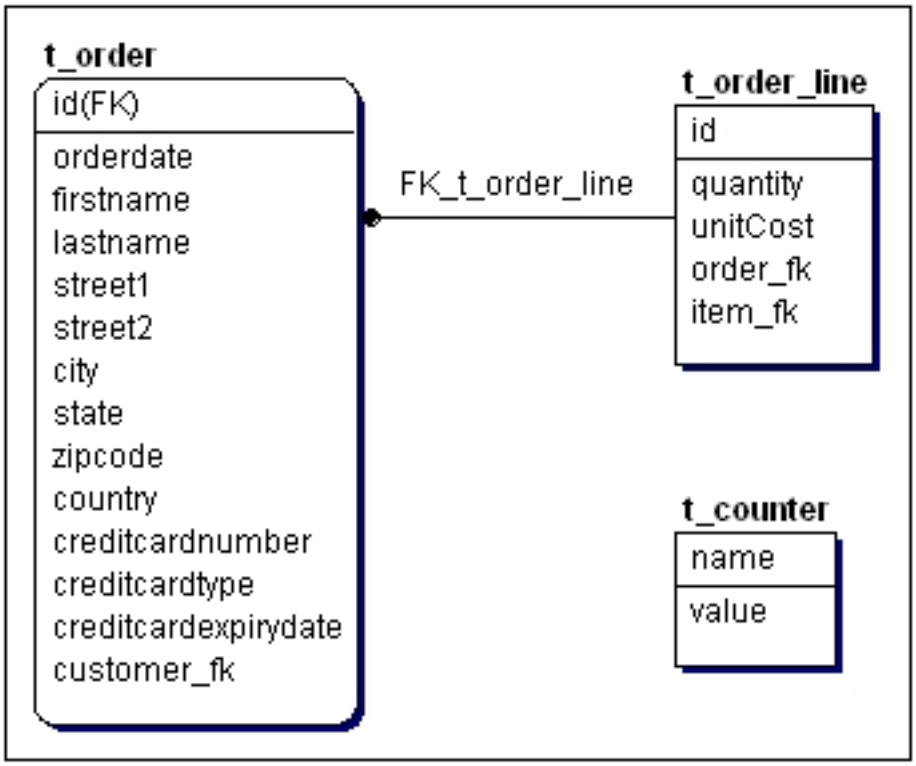


Figure 12 - Ajout de trois tables dans la base de données

Vue déploiement

Les applications PetStore Customer et PetStore Catalog restent respectivement déployées sur les postes de Bill et John. PetStore Order, quant à elle, est déployée sur plusieurs postes d'employés. Le tout, utilise la base de données distante MySQL via JDBC.

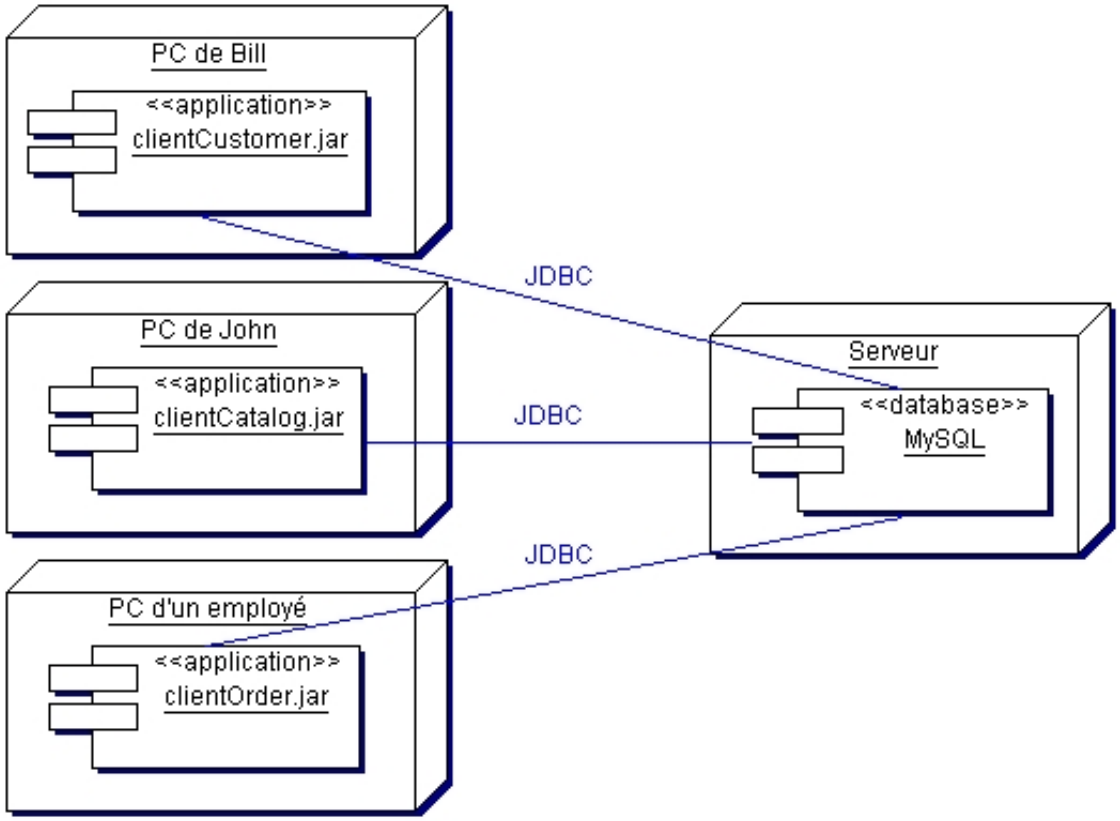


Figure 13 - Diagramme de déploiement des postes utilisateurs et de la base de données

Tout comme la version précédente de l'application, il y a un fichier .jar différent par type d'interface utilisateurs (clientCatalog.jar, clientCustomer.jar et clientOder.jar).

Implémentation

Vous pouvez maintenant développer l'application à partir de la version précédente, ou télécharger la liste des classes fournies pour commencer votre développement. Dans les documents fournis, vous avez la quasi-totalité des classes de l'interface graphique Petstore Order ainsi que tous les objets du domaine sauf Order et OrderLine. Vous aurez à développer la classe CustomerService en vous inspirant de CatalogService et OrderService.

Vous avez pu constater que les classes DAO se ressemblent très fortement. Les seules choses qui les différencient sont les ordres SQL d'accès aux tables. Pour factoriser un maximum de code dans la classe mère (AbstractDataAccessObject) on peut utiliser le design pattern Template Method.

Le design pattern Template Method permet de définir le squelette d'un algorithme et de déléguer certaines actions aux sous-classes.

Dans notre cas, la classe AbstractDataAccessObject définit le squelette suivant :

```
public
final void insert(
final DomainObject domainObject)
throws DuplicateKeyException {
Connection connection =
null;
Statement statement =
null;

try {
```

```
        // Gets a database connection
        connection = getConnection();
        statement = connection.createStatement();
        // Inserts a Row
        statement.executeUpdate(getInsertSqlStatement(domainObject));
}
catch (SQLException e) {
    // ...
    }

protected
abstract
String getInsertSqlStatement(DomainObject object);
}
```

La méthode getInsertSqlStatement() est abstraite et doit donc être implémentée par les sous classes de AbstractDataAccessObject pour retourner le bon ordre SQL. Ainsi, dans CategoryDAO :

```
final class CategoryDAO
extends AbstractDataAccessObject {

protected
String getInsertSqlStatement(
final DomainObject object) {

final Category category = (Category) object;

final
String sql;
        sql =
"INSERT INTO " + TABLE +
"(" + COLUMNS +
") VALUES ('" + category.getId() +
"', '" + category.getName() +
"', '" + category.getDescription() +
"' )";

return sql;
    }
```

Les classes MenuCatalog et MenuCustomer ont été remaniées pour isoler plusieurs méthodes identiques ayant été reportées dans la super-classe AbstractTextMenu. Plusieurs classes Swing vous sont fournies telles que le menu principal, deux écrans de liste et l'écran vous permettant de créer une commande.

Les classes UniqueIdGenerator et UniqueIdGeneratorDAO vous permettront d'obtenir des identifiants uniques.

Recette utilisateur

Téléchargez les classes de test représentant la recette utilisateur et exécutez la classe AllTests qui fait appel à toutes les autres classes. Pour tester les objets métiers liés à la commande, les classes OrderDAOTest et OrderLineDAOTest ont été développées sur le même modèle que les classes existantes CustomerDAOTest, CategoryDAOTest, ProductDAOTest ou ItemDAOTest.

Les tests de recette doivent prendre en compte toutes les couches. C'est pour cela que vous trouverez les trois nouvelles classes CatalogServiceTest, CustomerServiceTest et OrderServiceTest qui permettent de tester les trois façades. Par exemple, ci-dessous la méthode testFindAllCustomers de la classe CustomerServiceTest.

```
public void testFindAllCustomers()
throws Exception {

final
String id = getStringId();
    // First findAll

final
int firstSize = findAllCustomers(); // (1)
    // Creates an object
```

```
    createCustomer(id); // (2)
    // Ensures that the object exists

try {
        findCustomer(id);
    }
catch (ObjectNotFoundException e) {
        fail(
"
Object has been created it should be found");
    }
    // Second findAll

final
int secondSize = findAllCustomers(); // (3)
    // Checks that the collection size has increase of one (4)

if (firstSize + 1 != secondSize) fail(
"The collection size should have increased by 1");
    // Cleans the test environment
    deleteCustomer(id);  // (5)

try {
        findCustomer(id);
        fail(
"
Object has been deleted it shouldn't be found");
    }
catch (ObjectNotFoundException e) {
    }
}
```

Ce test effectue un premier appel à la méthode findAll (1) de la façade. Ceci lui indique le nombre de clients présents dans le système. Ensuite, il crée un nouveau client (2), refait un nouvel appel à la méthode findAll (3) et s'assure que la liste des clients a augmenté (4). Pour laisser le système dans l'état de départ, on supprime le client que l'on vient de créer (5).

Résumé

Cette version apporte deux évolutions majeures. Tout d'abord l'extension des objets métier avec la possibilité de créer, rechercher et supprimer des commandes d'animaux. Ensuite, la création d'un nouveau type d'interface graphique. Grâce au design pattern Facade on arrive à découpler encore plus la couche de présentation de la couche métier en regroupant tout le code métier à un seul endroit.

En Java vous avez deux moyens principaux de créer des interfaces graphiques : AWT ou Swing. Swing permet de créer des écrans plus riches tout en restant plus simple d'affichage (lightweight). De plus, comme nous l'avons vu, cette technologie s'affranchit entièrement de la plateforme et peut prendre l'aspect que vous désirez (Windows, Metal ou Motif). Pour cela, la technologie Swing a été utilisée.

Références

The Swing Tutorial http://docs.oracle.com/javase/tutorial/uiswing/

The JFC Swing Tutorial: A Guide to Constructing GUIs, Second Edition Kathy Walrath, Mary Campione, Alison Huml, Sharon Zakhour. Addison-Wesley. 2004.

Activity Diagram in UML http://www.developer.com/design/article.php/2247041

Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML, 2nd Edition, Volume 1 Mark Grand. Wiley. 2002.

Facade Pattern http://en.wikipedia.org/wiki/Facade_pattern

Template Method Pattern http://en.wikipedia.org/wiki/Template_method_pattern