

GEST-D-201 – Informatique pour la gestion II

Semaine intensive 2006-2007

Prof. : Hugues Bersini

18-23/12/2006

Objectifs

L'objectif du cours est de vous familiariser avec la démarche du développement informatique et de ses architectures de base : les bases de données, la démarche algorithmique, la notion de programme informatique et les mécanismes clefs de la programmation orientée objet.

Projet

Durant cette semaine intensive, vous aurez à développer une petite application destinée à gérer les réservations aux différentes représentations d'une salle de spectacles. Votre programme interagira avec une base de données pour en extraire les informations et y stocker les réservations.



Le projet s'organise en plusieurs étapes et les notions requises vous seront exposées au fur et à mesure :

A réaliser	Théorie liée	Objectif
Schéma relationnel de la BD	Rappel : les bases de données relationnelles	Lundi AM
Base de données	MS Access (Tour d'horizon)	Lundi Midi
Diagramme de classes	Programmation OO et UML	Mardi Soir
Application v1 : OO	Programmation OO en Java	Mercredi Soir
Requêtes DB	Interrogation de BD : Langage SQL	Jeudi Midi
Application v2 : JDBC	Connexions aux BD en Java avec JDBC	Vendredi AM
Application v3 : SWING	Interfaces graphiques Java avec SWING	Samedi Midi

Enoncé

Vous devez réaliser un programme s'occupant de la gestion des réservations d'une salle de spectacle. Votre programme vend des réservations pour une représentation (un certain jour à une certaine heure) d'un spectacle (caractérisé au minimum par son auteur, son titre et son type). Un client, identifié par ses nom, prénom, adresse et numéro de téléphone, peut effectuer autant de réservations qu'il le souhaite et peut-être ou non abonné. Si le client n'est pas abonné, il paie ses réservations par débit de son porte-monnaie électronique. Si le client est un abonné, le système conserve l'année de son inscription, il bénéficie d'une priorité sur les réservations (il peut réserver des places une ou deux semaines avant les non abonnés) et il achète ses places à crédit (il paie l'ensemble de ses réservations à la fin de l'année). Une fois la réservation enregistrée, le programme permet de sélectionner les places désirées par le client et un ticket est émis pour chaque place réservée.

1. Schéma relationnel

Tracez le schéma relationnel de la base de données nécessaire pour stocker et gérer les données du programme :

1. Combien de tables sont-elles nécessaires ?
2. Quelles sont ces tables ?
3. Quels champs contiennent ces tables ?
4. De quel type sont-ils ?
5. Quelles tables sont reliées entre elles ?
6. Quel est le type de ces relations ?
7. Quelles seront les clés primaires et étrangères ?

2. Base de données (Microsoft Access)

Réalisez la base de données correspondant à votre schéma relationnel en Microsoft Access :

1. Créez les différentes tables
2. Créez les champs (colonnes) des tables en précisant leur type
3. Définissez les clés primaires (veillez à ce que les ID soient attribués automatiquement par la base de données)
4. Définissez les relations entre classes en forçant l'intégrité référentielle (double-cliquez sur une relation et cochez les cases requises (cf. encadré))
5. Introduisez quelques données dans les différentes tables

Encadré 1 : Créer une base de données Microsoft Access

Lancez Microsoft Access, créez une nouvelle base de données vierge et définissez vos tables (en mode conception (« Design »)) puis vos relations. Les relations peuvent être tracées au départ du menu « Outils » puis « Relations » (Tools > Relationships). Pour introduire des données dans une table, double-cliquez sur la table choisie et introduisez les données dans la table. Naturellement, dans le cas de tables liées par une relation « 1 à n », il faut introduire les données dans la table du côté « 1 » avant la table du côté « n ».

3. Diagramme de classes

Réalisez le diagramme de classe de votre future application :

1. Combien de classes sont-elles nécessaires ?
2. Quelles sont ces classes ?
3. Quelles classes sont reliées entre elles ?
4. Quel est le type de ces relations ?
5. Quels seront les attributs de ces classes ?
6. Quelles seront les méthodes de chacune de ces classes ?

4. Application V1 – OO (Java avec Eclipse)

Développez une première version de ce programme correspondant à votre diagramme de classes. Votre programme communiquera avec l'utilisateur via la console de commandes.


1. Créez les classes correspondant à votre diagramme
2. Implémentez les relations entre classes correspondant à votre diagramme
3. Créez une classe principale contenant une méthode « main »
4. Dans cette méthode « main », instanciez quelques objets de chaque classe, créez quelques réservations, et affichez tous les objets créés sur la console.
5. Lancez votre programme pour le tester (cf. ci-dessous)
6. Enrichissez ensuite votre méthode « main » pour qu'elle stocke les différents objets créés dans une collection « ArrayList ».
7. Ordonnez l'affichage sur la console de tous les objets présents dans la collection « ArrayList » en veillant à exploiter au maximum le polymorphisme.
8. Lancez à nouveau votre programme pour vous assurer qu'il fonctionne.

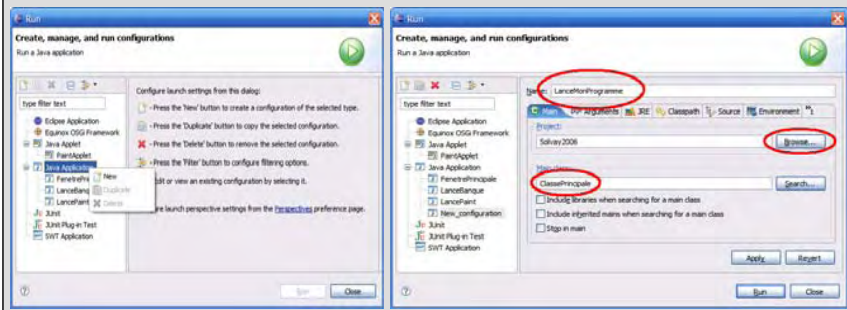
Encadré 2 : Exécuter une application Java

Pour lancer un programme Java, il faut normalement :

1. le compiler (le traduire en « bytecode »),
2. lancer la machine virtuelle Java (JVM),
3. et demander à la machine virtuelle Java de lancer la classe principale (celle qui contient le « main »).

Or il se trouve que l'environnement Eclipse est conçu pour compiler vos classes au fur et à mesure que vous les écrivez (d'où la détection en temps réel de vos erreurs de syntaxe). Vous ne devez donc pas vous soucier de la 1^{ère} étape. Quant aux 2 étapes suivantes, elles peuvent être réalisées en une seule par Eclipse, à condition de lui indiquer quel est le nom de la classe principale contenant le main. Pour ce faire, suivez les étapes suivantes (à réaliser une fois pour toutes) :

1. Dans le menu « Run », cliquez sur l'option « Run... »
2. Cliquez avec le bouton droit sur « Java Application » puis « New »
3. Donnez un nom à la configuration, indiquez celui de votre projet (si ce n'est fait) et de votre classe principale, puis cliquez sur le bouton Run.
4. Désormais, pour démarrer, il suffira de cliquer sur  dans la barre d'outils



5. Requêtes SQL (Microsoft Access)

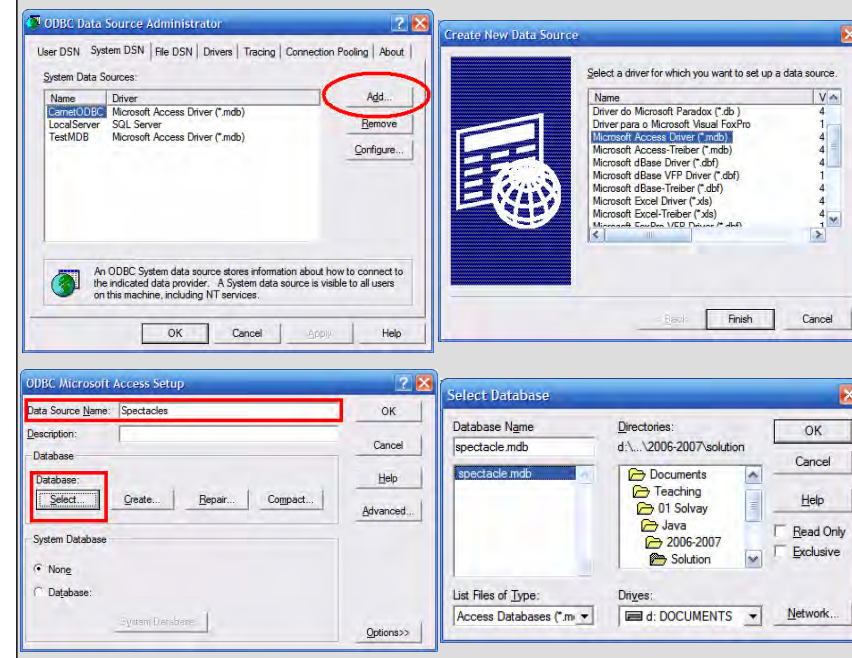
Ouvrez à nouveau votre base de données Access et créez les quelques requêtes suivantes (vous pouvez utiliser le concepteur de requêtes de Access, mais veillez dans ce cas à bien examiner le code SQL généré) :

- Lister tous les clients par ordre alphabétique
- Lister toutes les représentations avec le nom des spectacles correspondants
- Lister toutes les réservations avec le nom du client pour un spectacle donné
- Insérer une nouvelle réservation
- Modifier le statut « Payé » d'une réservation en particulier
- Supprimer tous les tickets d'une réservation précise

Une fois les requêtes créées, déclarez votre BD comme « DataSource ODBC » (cf. encadré)

Encadré 3 : Déclarer une base de données comme Source ODBC

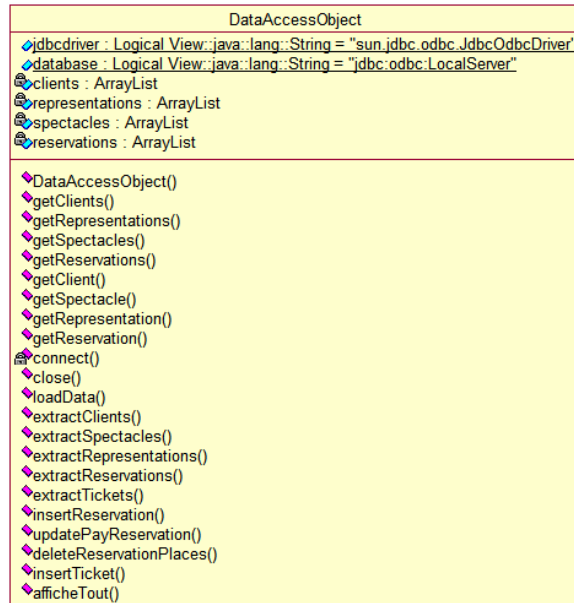
Pour rendre votre BD accessible par des programmes extérieurs, il faut la déclarer comme source de données ODBC, via le panneau de contrôle ODBC, onglet « User DSN » :



6. Application V2 - JDBC

L'objectif de cette étape est de connecter votre programme à votre BD pour qu'il en extraie les informations et puisse y ajouter ou en modifier des données. A la fin de cette étape, votre programme devra pouvoir afficher le contenu de la BD à l'écran et lui envoyer des actions :

1. Créez une nouvelle classe « DataObjecAccess » qui sera chargée de communiquer avec la base de données. Le diagramme UML de cette classe vous est proposé ci-dessous (remplacez « LocalServer » par le nom de votre Source DSN ODBC):



Encadré 4 : Squelette de la classe DataAccessObject

La structure et le code de base de cette classe vous sont fournis ci-dessous. Veuillez à reproduire ce code tel quel avant d'implémenter les attributs et les méthodes du diagramme :

```
import java.sql.*;
import java.util.*;
public class DataAccessObject {
    public static final String jdbcdriver="sun.jdbc.odbc.JdbcOdbcDriver";
    public static final String database = "jdbc:odbc:NOMDEVOTREDSNODBC";
    private Connection conn;
    public DataAccessObject(){
        try{
            Class.forName(jdbcdriver);
        } catch(Exception e){e.printStackTrace();}
    }
    private void connect() throws SQLException{
        conn = DriverManager.getConnection(database);
    }
    public void close() throws SQLException{
        if(conn!=null)
            conn.close();
    }
}
```

Encadré 5 : Envoyer une requête action à la base de données

Pour envoyer une requête action à la base de données à l'intérieur de vos méthodes, le code de base est le suivant :

```
try {
    if(conn==null)
        this.connect();
    Statement stmt = conn.createStatement();
    String query = "DELETE * FROM tickets WHERE ticket_id = XXX;";
    stmt.executeUpdate(query);
    stmt.close();
} catch(Exception e){
    e.printStackTrace();
}
```

Encadré 6 : Extraire des données de la base de données

Pour extraire des données de la base de données à l'intérieur de vos méthodes, le code de base est le suivant :

```
try {
    if(conn==null)
        this.connect();
    Statement stmt = conn.createStatement();
    String query = "SELECT * FROM clients ORDER BY client_name";
    ResultSet rs = stmt.executeQuery(query);
    while(rs.next()){
        int c_id = rs.getInt("client_id");
        String c_n = rs.getString("client_name");
        String c_fn = rs.getString("client_firstname");
        String c_ad = rs.getString("client_address");
        String c_ph = rs.getString("client_phone");
        int c_ab = rs.getInt("client_abonne");
        double c_c = rs.getDouble("client_credit");
        int c_am = rs.getInt("client_anneemembre");
        clients.add(new Client(c_id,c_n,c_fn,c_ad,c_ph,c_c,c_am));
    }
    rs.close();
    stmt.close();
} catch(Exception e){
    e.printStackTrace();
}
```

2. Une fois cette nouvelle classe réalisée, vous remplacerez le code de votre ancienne méthode « main » (qui créait des objets artificiellement) par un code permettant de récupérer les objets de la classe « DataAccessObject » ci-dessus et vous demanderez à tous ces objets de s'afficher à l'écran sur la console. De cette façon, votre programme, en se lançant, chargera toutes les données de la BD et fera défiler tout son contenu sur la console. Vous y appellerez également une fois chacune des 4 méthodes transactionnelles (celles qui modifient des données).

Classe DataAccessObject

```
import java.sql.*;
import java.util.*;
public class DataAccessObject {
    public static final String jdbcdriver="sun.jdbc.odbc.JdbcOdbcDriver";
    public static final String database = "jdbc:odbc:LocalServer";
    private Connection conn;

    // A FAIRE PAR VOUS: Déclarer les ArrayList pour stocker les objets
    // (clients, spectacles, etc.)

    public DataAccessObject(){
        try{
            Class.forName(jdbcdriver);
        } catch(Exception e){e.printStackTrace();}

        // A FAIRE PAR VOUS: Initialiser les ArrayList
    }

    private void connect() throws SQLException{
        conn = DriverManager.getConnection(database);
    }

    public void close() throws SQLException{
        if(conn!=null)
            conn.close();
    }

    // A FAIRE PAR VOUS: Définir des méthodes pour renvoyer les ArrayList

    // A FAIRE PAR VOUS: Définir des méthodes pour extraire les données
    // de chaque type de la base de données, créer les objets
    // correspondants et les stocker dans les ArrayList

    // A FAIRE PAR VOUS: Définir des méthodes pour créer une nouvelle
    // réservation dans la BD, pour supprimer les tickets d'une
    // réservation en particulier, pour enregistrer le paiement d'une
    // réservation donnée et pour créer les tickets d'une réservation.
}
```

7. Application V3 - SWING

Maintenant que votre programme fonctionne et qu'il permet de consulter la BD et d'interagir avec elle, il ne vous reste plus qu'à créer une petite interface graphique qui remplacera la simple console et permettra aux utilisateurs de consulter la BD plus facilement et d'enregistrer des réservations. Votre application devra permettre :

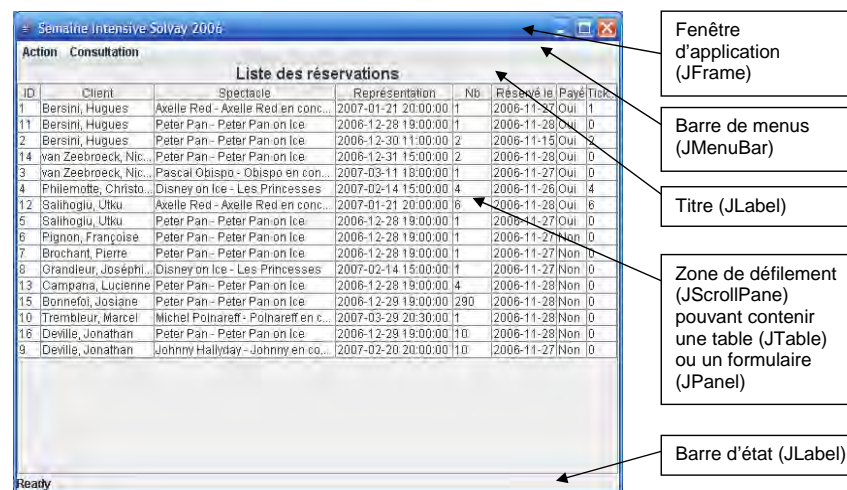
- de lister les clients et les réservations existantes
- d'enregistrer une nouvelle réservation (en tenant compte des dates de priorité):
 - Choisir client, représentation et nombre de places dans des listes
 - Sélectionner les places (et puis création des tickets)
 - Paiement de la réservation
 - par débit du porte-monnaie pour les non abonnés
 - par crédit pour les abonnés

Pour réaliser cela, vous devrez mettre en œuvre différents mécanismes propres aux interfaces graphiques en Java, dites SWING, ce que vous ferez en 3 étapes.

7.1. Affichage

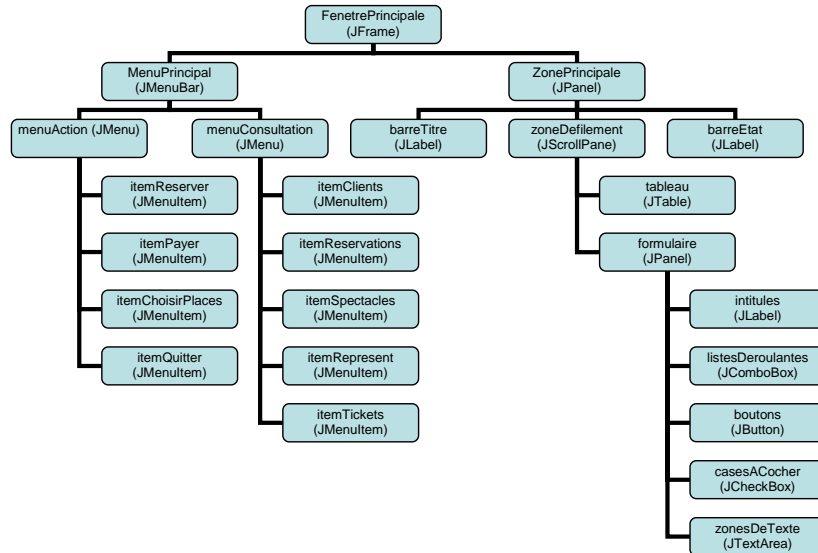
La première étape consistera à composer la partie visible de votre interface graphique. Il vous faudra donc commencer par créer une fenêtre d'application dans laquelle les différents éléments de votre interface viendront se placer. A ce stade, il ne sera pas encore question ni d'interactions avec l'utilisateur (l'utilisateur ne peut encore rien « faire » avec le programme) ni d'interactions avec la base de données ou avec les autres classes (aucun de nos objets n'y intervient encore). Autrement dit, vous devez créer l'interface graphique dans cette première étape indépendamment du reste de votre programme déjà écrit. Nous ne reconnecterons les deux que dans la troisième et dernière étape.

Votre interface graphique, qui pourrait ressembler à celle de la première page, comportera les éléments suivants :



Vous constaterez donc qu'il y a deux sortes de composants graphiques à créer et à gérer : ceux qui peuvent eux-mêmes contenir des composants et les autres. On appelle les premiers « Container » et les seconds « Component ». Notez que tous les « Container » sont eux-mêmes des « Component ».

L'organisation de votre interface sera donc la suivante :



Pour mettre ceci en œuvre, vous devrez donc créer 3 classes successivement :

1. Classe FenetrePrincipale héritant de JFrame (code intégral fourni en annexe)
2. Classe MenuPrincipal héritant de JMenuBar (code de base fourni en annexe)
3. Classe ZonePrincipale héritant de JPanel (code de base fourni en annexe)

La première incorporera les deux autres, et tous les autres composants viendront se loger à l'intérieur de ces dernières.

Encadré 7 : Exploiter les composants graphiques Java SWING prédéfinis

Conteneurs ou non, il y a deux façons d'exploiter les composants graphiques en Java :

- En redéfinissant (par héritage) la classe fournie par le langage (ce qu'on fait souvent pour des conteneurs, plus rarement pour les autres composants) :

```

import javax.swing.JFrame;
public class FenetrePrincipale extends JFrame {
    public FenetrePrincipale() {
        this.setSize(800,600);
        this.setTitle("Nom du programme");
        this.setVisible(true);
    }
}
  
```

C'est ce que vous devrez faire pour la fenêtre principale, pour la barre de menus et pour la zone principale (vos 3 classes à créer).

- En créant simplement une instance de la classe (un objet) dont on modifiera les paramètres qui nous intéressent :

- Exemple pour un intitulé (« étiquette »)
 - Avant la déclaration de la classe :


```
import javax.swing.JLabel;
```
 - Parmi les déclarations d'attributs :


```
private JLabel title;
```
 - Dans le constructeur :


```
title = new JLabel("Welcome", JLabel.CENTER);
```
 - Dans d'autres méthodes (pour changer le contenu du texte) :


```
title.setText("NOUVEAU TEXTE");
```

Encadré 7 : Exploiter les composants graphiques Java SWING prédéfinis (suite)

- Exemple pour un bouton :
 - Avant la déclaration de la classe :


```
import javax.swing.JButton;
```
 - Dans les variables membres :


```
private JButton bouton;
```
 - Dans le constructeur :


```
bouton = new JButton("Texte sur le bouton");
```
- Exemple pour une liste déroulante :
 - Avant la déclaration de la classe :


```
import javax.swing.JComboBox;
```
 - Dans les variables membres :


```
private JComboBox liste;
```
 - Dans le constructeur :


```
liste = new JComboBox();
liste.addItem("1er élément de la liste");
liste.addItem("2ème élément de la liste");
liste.addItem("3ème élément de la liste");
...
```

Encadré 8 : Faire apparaître les composants à l'écran

Une fois les conteneurs et composants déclarés et instanciés, il reste à les faire apparaître à l'écran. Le code ci-dessus (cf. Fenêtre Principale) permettra de faire apparaître la fenêtre d'application à l'écran (« this.setVisible(true) »). Il ne reste donc plus qu'à modifier votre méthode « main » pour qu'elle instancie désormais la classe FenetrePrincipale :

```

public static void main(String[] args) {
    new FenetrePrincipale();
}
  
```

Il reste enfin à incorporer dans la fenêtre les différents composants requis. Pour ce faire, tous les conteneurs possèdent une méthode « add(Component c) » qui leur permet d'incorporer et d'afficher un nouveau composant. Cela se fait habituellement dans le constructeur du conteneur. Si l'on reprend donc le code de la fenêtre principale, cela donne :

```

import javax.swing.JFrame;
public class FenetrePrincipale extends JFrame {
    private MenuPrincipal menu;
    private ZonePrincipale zone;
    public FenetrePrincipale() {
        this.setSize(800,600);
        this.setTitle("Nom du programme");
        zone = new ZonePrincipale();
        menu = new MenuPrincipal();
        this.setJMenuBar(menu);
        this.add(zone);
        this.addWindowListener(this);
        this.setVisible(true);
    }
}
  
```

Vous noterez cependant que si l'on utilise bien « add(zone) » pour incorporer la zone principale dans la fenêtre, on utilise en revanche une méthode spécifique « setJMenuBar(menu) » pour mettre en place la barre de menu. C'est l'une des deux seules exceptions que vous rencontrerez. De fait, pour incorporer par exemple les éléments de menus à l'intérieur des menus et ceux-ci à leur tour à l'intérieur de la barre de menus :

Encadré 8 : Faire apparaître les composants à l'écran (suite)

```
private JMenu actions;
private JMenuItem clients;
public MenuPrincipal(){
    actions = new JMenu("Action");
    clients = new JMenuItem("Liste des clients");
    affichage.add(clients);
    this.add(actions);
}
}
```

L'autre exception concernera l'inclusion alternative du tableau ou des écrans d'action (prise d'une réservation, choix des places, paiement) dans la zone de défilement. En effet, pour créer et afficher le tableau dans la zone en question, il vous faudra utiliser le code suivant dans la classe ZonePrincipale :

- Avant la déclaration de la classe :

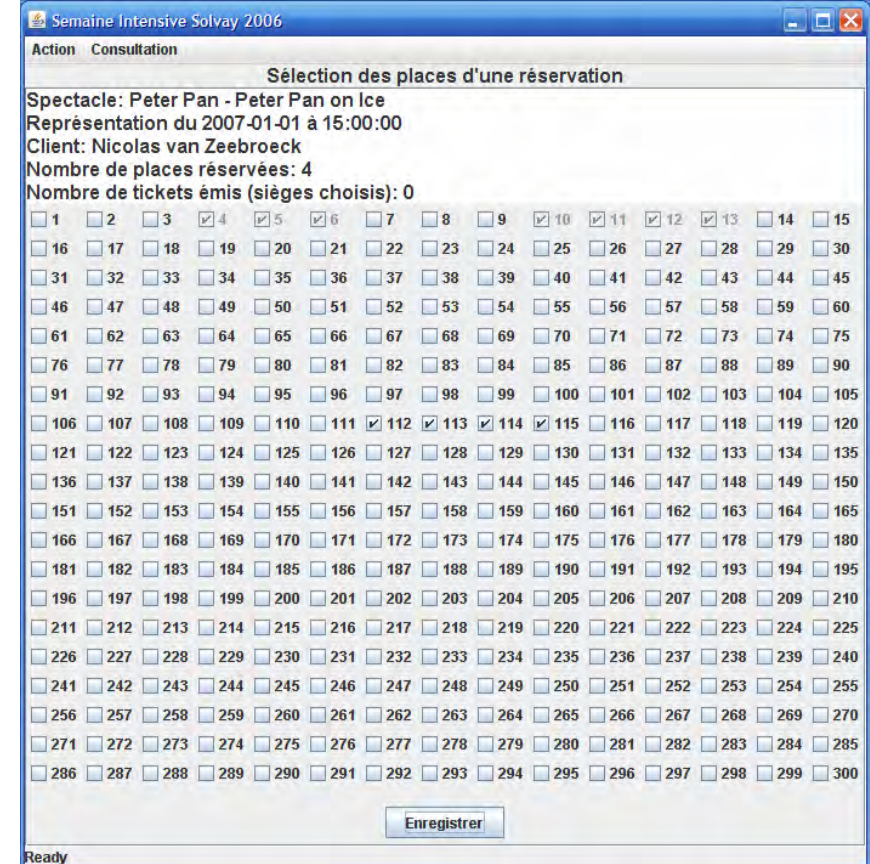
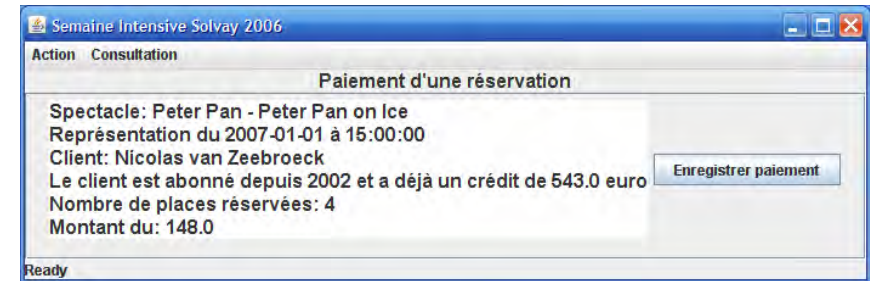
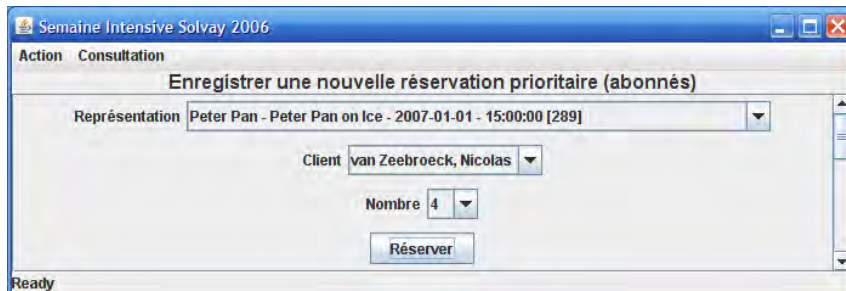

```
import javax.swing.JTable;
import javax.swing.JScrollPane;
```
- Dans les variables membres :


```
private JScrollPane zoneDefilement;
private JTable tableau;
```
- Dans le constructeur :


```
zoneDefilement = new JScrollPane();
this.add(zoneDefilement, "Center");
```
- Pour créer et incorporer le tableau dans la zone de défilement (avec des données arbitraires) :


```
String[][] donnees = {{ "R1C1", "R1C2"}, {"R2C1", "R2C2"},
                       {"R3C1", "R3C2"} };
String[] titres = {"TitreC1", "TitreC2"};
tableau = new JTable(donnees, titres);
zoneDefilement.setViewportView(table);
```

A titre d'exemple, voici à quoi pourraient ressembler les écrans de votre application (notez toutefois qu'à ce stade les listes et écrans n'afficheront encore aucune donnée de la base de données) :



7.2. Interactivité (gestion des événements)

Maintenant que la partie visible de votre application est en place, il vous faut permettre à l'utilisateur d'interagir avec votre programme. Pour cela, il va falloir mettre en œuvre des mécanismes permettant de traiter (gérer) les actions de l'utilisateur. En Java, c'est ce qu'on appelle la « gestion des événements », les « événements » s'entendant comme les actions initiées par l'utilisateur.

Dans votre cas, 3 types d'événements (ou actions) doivent être traités :

- La fermeture de la fenêtre (pour qu'elle entraîne la fin du programme)
- Les sélections dans les menus et les clics sur les boutons (pour déclencher les actions requises)
- La sélection d'une ligne dans la table (pour savoir quelle réservation traiter)

La prise en charge de ces différents événements suppose qu'un objet précis soit responsable de la détection des actions de l'utilisateur pour chaque type d'événement concerné et du déclenchement des traitements correspondants. Pour cela, l'objet dit « écouteur » (Listener) doit donner aux composants source des événements la garantie qu'il est capable de prendre en charge les événements du type concerné et se faire connaître auprès d'eux :

1. Choisir la classe à définir comme écouteur pour chaque type d'événement
2. Implémenter l'interface d'écoute correspondante dans la classe choisie
 - « WindowListener » pour la fermeture de la fenêtre
 - « ActionListener » pour la gestion des menus et boutons
 - « ListSelectionListener » pour la gestion des sélections dans la table
3. Instancier la classe « écouteur » et l'associer à la source d'événements

Classe Source	Événements	Interface	Classe Ecouteur
FenetrePrincipale	Fermeture de la fenêtre	WindowListener	FenetrePrincipale
JMenuItem	Choix dans un menu	ActionListener	GereMenus
JButton	Clic sur un bouton	ActionListener	GereMenus
JTable	Sélection d'une ligne	ListSelectionListener	GereTables

Vous devrez donc :

1. Rendre la classe FenetrePrincipale capable de se gérer elle-même
2. Créer 2 classes d'écouteurs spécifiques : GereMenus, GereTables
3. Instancier ces 2 classes
4. Associer les écouteurs à la source d'événements qu'ils doivent gérer

Chaque fois que vous aurez programmé un écouteur et que vous l'aurez associé à sa source, relancez votre programme pour tester le bon fonctionnement de l'interface (les méthodes d'actions pourraient dans un premier temps se contenter d'afficher un message sur la console). Notez que le code des 3 classes d'écouteurs (GereMenus, GereListes et GereTables) vous est fourni tel quel en annexe. Il faudra naturellement l'adapter au besoin à votre propre application (notamment pour appeler les bonnes méthodes en cas d'événement).

Encadré 9 : Implémenter une interface de gestion événementielle

Pour implémenter une interface, il faut et il suffit de le préciser dans la déclaration de la classe avec la clause « implements » :

```
- public class FenetrePrincipale extends JFrame implements WindowListener{
- public class GereMenus implements ActionListener{
- public class GereTables implements ListSelectionListener{
```

et de redéfinir la ou les méthode(s) correspondant à chaque événement possible :

```
- Pour la gestion de la fenêtre (WindowListener) :
    public void windowActivated(WindowEvent arg0) {}
    public void windowClosed(WindowEvent arg0) {}
    public void windowClosing(WindowEvent arg0) {
        System.exit(0);
    }
    public void windowDeactivated(WindowEvent arg0) {}
    public void windowDeiconified(WindowEvent arg0) {}
    public void windowIconified(WindowEvent arg0) {}
    public void windowOpened(WindowEvent arg0) {}

- Pour la gestion des menus (ActionListener) :
    public void actionPerformed(ActionEvent e) {...}

- Pour la gestion des menus (ListSelectionListener) :
    public void valueChanged(ListSelectionEvent e) {
        if (e.getValueIsAdjusting())
            return;
        ListSelectionModel lsm;
        lsm = (ListSelectionModel) e.getSource();
        if(lsm.isSelectionEmpty()){
            // Que faire si aucune ligne n'a été sélectionnée?
        } else {
            // La ligne sélectionnée est :
            int i = lsm.getMinSelectionIndex();
            // Que faire avec cette information ?
        }
    }
}
```

Encadré 10 : Associer une source et son écouteur

En implémentant ce qui précède dans la classe concernée, on rend alors la classe en question capable de prendre en charge les événements du type correspondant. Il reste toutefois à associer chaque source potentielle d'événements (la fenêtre principale, chaque élément de menu, chaque bouton, chaque liste déroulante et chaque table) à son objet écouteur, instance de la classe correspondante. Cela se fait en général pendant ou juste après la construction de l'objet source au moyen d'une méthode « addXXXXXXListener(objetEcouteur) » :

```
- Pour informer la fenêtre qu'elle est son propre écouteur (dans le constructeur)
    this.addWindowListener(this);

- Pour associer un élément de menu à son écouteur :
    clients = new JMenuItem("Liste des clients");
    clients.addActionListener(new GereMenus());

- Pour associer un bouton à son écouteur :
    bouton = new JButton("Sauver la réservation");
    bouton.addActionListener(new GereMenus());

- Pour associer une table à son écouteur :
    table = new JTable(dataTable,headersTable);
    table.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    table.getSelectionModel().addListSelectionListener(new
        GereTables());
```

7.3. Relier votre interface à votre modèle orienté objet

Maintenant que la partie visible de votre application est en place et qu'elle est capable de prendre en charge les différentes actions de l'utilisateur, le moment est venu d'incorporer votre application précédente (gestion des réservations) à l'intérieur de votre interface graphique. Autrement dit, il va falloir « connecter » votre interface graphique à vos objets clients, spectacles, représentations, réservations et tickets, afin d'extraire les données nécessaires à l'affichage des tableaux ou formulaires et de pouvoir envoyer les nouvelles informations vers la base de données.

a) affichage des données dans le tableau

Pour cela, il faudra tout d'abord que votre interface graphique (et en particulier votre zone principale) puisse recueillir les données provenant de la base de données. Typiquement, cela sera nécessaire pour afficher les données requises dans un tableau.

Pour cela, il faudra :

1. Récupérer les données de la BD sous forme d'objets. Autrement dit, interroger l'objet de type `DataAccessObject` pour qu'il renvoie la collection des clients ou des réservations, etc :

```
ArrayList data = dao.getClients();
```
2. Demander à chacun des objets (clients, réservations, etc.) de renvoyer ses informations de façon structurée, sous forme d'un vecteur à « [] ». Autrement dit, il faudra boucler sur l'ArrayList renvoyée par le `DataAccessObject` et demander à chaque élément de l'ArrayList de renvoyer ses données :

```
for(int i=0;i<data.size();i++){
    dataTable[i] = ((DataObject) data.get(i)).getDataArray();
}
```

Notez que cela suppose que tous vos objets (clients, spectacles, etc.) possèdent un message (méthode) de type :

```
public Object[] getDataArray(){
    ...
}
```

3. Créer la table en lui envoyant ces données

```
table = new JTable(dataTable,headersTable)
```

Notez que « headersTable » devrait désigner un tableau de « String » comprenant les intitulés des différentes colonnes.
4. Afficher la table à l'intérieur de la zone de défilement

```
zoneDefilement.setViewportViewView(table)
```

b) Créer/modifier des données dans la BD (nouvelle réservation, paiement, choix de places)

Dans votre classe `ZonePrincipale`, une méthode correspondra à chaque type d'opération possible et sera appelée lorsque le bouton d'enregistrement du formulaire sera activé. Ces méthodes récupéreront les informations du formulaire (valeurs choisies dans les listes, cases cochées, etc.) et les vérifieront (sont-elles valables, reste-t-il suffisamment de places disponibles pour la représentation choisie, etc.) Enfin, la méthode enverra un message à la classe `DataAccessObjects` pour appeler (comme le faisait votre « main » avant la création de l'interface graphique) la méthode correspondant à l'opération, cette dernière ayant pour mission d'effectuer les modifications dans la base de données.

Classe FenetrePrincipale (code complet)

```
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import javax.swing.JFrame;
public class FenetrePrincipale extends JFrame implements WindowListener{
    private MenuPrincipal menu;
    private ZonePrincipale zone;
    public FenetrePrincipale(){
        this.setSize(800,720);
        this.setTitle("Semaine Intensive Solvay 2006");
        zone = new ZonePrincipale();
        menu = new MenuPrincipal(zone);
        this.setJMenuBar(menu);
        this.add(zone);
        this.addWindowListener(this);
        this.setVisible(true);
    }
    public void windowActivated(WindowEvent arg0) {}
    public void windowClosed(WindowEvent arg0) {}
    public void windowClosing(WindowEvent arg0) {
        zone.closeDAOConnection();System.exit(0);
    }
    public void windowDeactivated(WindowEvent arg0) {}
    public void windowDeiconified(WindowEvent arg0) {}
    public void windowIconified(WindowEvent arg0) {}
    public void windowOpened(WindowEvent arg0) {}
    public static void main(String[] args){
        new FenetrePrincipale();
    }
}
```

Classe MenuPrincipale

```

import java.awt.event.ActionListener;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
public class MenuPrincipale extends JMenuBar {
    private JMenu actions, affichage;

    // A FAIRE PAR VOUS: Déclarer les JMenuItem's

    public MenuPrincipale(ZonePrincipale z){
        actions = new JMenu("Action");
        affichage = new JMenu("Consultation");

        clients = new JMenuItem("Liste des clients");
        clients.addActionListener(new GereMenus(GereMenus.CLIENTS, z));

        // A FAIRE PAR VOUS: Idem pour les autres JMenuItem's

        affichage.add(clients);

        // A FAIRE PAR VOUS: Idem pour les autres JMenuItem's

        this.add(actions);
        this.add(affichage);
    }
}

```

Classe ZonePrincipale

```

import java.awt.*;
import java.sql.SQLException;
import java.util.ArrayList;
import javax.swing.*;
public class ZonePrincipale extends JPanel {
    private DataAccessObject dao;
    private JTable table;
    private JScrollPane zoneDefilement;
    private JPanel formulaire;
    private JLabel title, etat;

    // A FAIRE PAR VOUS: Déclarer les autres attributs

    public ZonePrincipale(){
        this.setLayout(new BorderLayout());
        dao = new DataAccessObject();
        title = new JLabel("Welcome",JLabel.CENTER);
        scrolling = new JScrollPane();
        etat = new JLabel("Ready");

        // A FAIRE PAR VOUS: Initialiser les autres attributs

        this.add(title,"North");
        this.add(scrolling,"Center");
        this.add(etat,"South");
    }

    // A FAIRE PAR VOUS: Méthode(s) à appeler pour afficher la table
    // A l'intérieur de cette méthode (ex. pour les clients):

    ArrayList data = dao.getClients();
    dataTable = new Object[data.size()][Client.HEADERS.length];
    headersTable = Client.HEADERS;
    for(int i=0;i<data.size();i++){
        dataTable[i] = ((DataObject)data.get(i)).getDataArray();
    }
    table = new JTable(dataTable,headersTable);
    table.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    table.getSelectionModel().addListSelectionListener(new
GereTables(this));
    scrolling.setViewportViewView(table);

    // A FAIRE PAR VOUS: Méthode(s) à appeler pour afficher les
    // formulaires possibles (réservation, paiement, choix de places)
    // A l'intérieur de ces méthodes au besoin:

    JComboBox lc = new JComboBox();
    lc.addItemListener(new GereListes(GereListes.CLIENTS, this));

    JButton b = new JButton("Réserver");
    b.addActionListener(new GereMenus(GereMenus.FORMSAVE, this));

    JTextArea status = new JTextArea();
    status.setText(s);
    status.setEditable(false);

    JCheckBox lp = new JCheckBox("Intitulé");
    lp.setSelected(true);
    lp[i].setEnabled(false);

    scrolling.setViewportViewView(formulaire);
}

```

```

// A FAIRE PAR VOUS: Méthode(s) à appeler si l'utilisateur
// sélectionne une ligne du tableau ou un élément dans une liste
// (pour stocker ce choix)

// A FAIRE PAR VOUS: Méthode(s) à appeler pour sauver
// les changements faits par l'utilisateur: sauver la réservation,
// sauver le paiement, sauver les places choisies
// A l'intérieur de ces méthodes, au besoin :

// Pour connaître le numéro de la ligne sélectionnée dans une liste :
int ligneSelectionnee = liste.getSelectedIndex();

// Pour vérifier si une case (JCheckBox) est sélectionnée/activée :
lp.isSelected(); // La case est-elle sélectionnée ?
lp.isEnabled(); // La case est-elle activée (=cochable) ?

public void closeDAOConnection(){
    try {
        this.dao.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

```

Classe GereTables (complète mais à adapter)

```

package spectacles;
import javax.swing.ListSelectionModel;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
public class GereTables implements ListSelectionListener{
    private ZonePrincipale zone;
    public GereTables(ZonePrincipale z){
        this.zone = z;
    }
    public void valueChanged(ListSelectionEvent e) {
        if (e.getValueIsAdjusting())
            return;
        ListSelectionModel lsm = (ListSelectionModel) e.getSource();
        if(lsm.isEmpty())
            zone.rowSelected(-1);
        else
            zone.rowSelected(lsm.getMinSelectionIndex());
    }
}

```

Classe GereMenus (complète mais à adapter)

```

package spectacles;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class GereMenus implements ActionListener{
    public static final int QUITTER = 0;
    public static final int CLIENTS = 1;
    public static final int SPECTACLES = 2;
    public static final int REPRESENTATIONS = 3;
    public static final int RESERVATIONS = 4;
    public static final int TICKETS = 5;
    public static final int FORMRES = 6;
    public static final int FORMSAVE = 7;
    public static final int PPAY = 8;
    public static final int PLCONF = 9;
    public static final int FORMPAY = 10;
    public static final int FORMRESPRIO = 11;
    public static final int FORMRESPLACES = 12;
    private int ceQueJEcoute;
    private ZonePrincipale zone;
    public GereMenus(int ceQueJEcoute, ZonePrincipale z){
        this.ceQueJEcoute = ceQueJEcoute;
        this.zone = z;
    }
    public void actionPerformed(ActionEvent e) {
        switch(this.ceQueJEcoute){
            case QUITTER: zone.closeDAOConnection(); System.exit(0);
            case CLIENTS: zone.afficheTable(ZonePrincipale.TCLIENTS);
            case SPECTACLES:
                zone.afficheTable(ZonePrincipale.TSPECTACLES); break;
            case REPRESENTATIONS:
                zone.afficheTable(ZonePrincipale.TREPRESENTATIONS); break;
            case RESERVATIONS:
                zone.afficheTable(ZonePrincipale.TRESERVATIONS); break;
            case TICKETS: zone.afficheTable(ZonePrincipale.TTICKETS);
            case FORMRES: zone.openFormReservation(false); break;
            case FORMSAVE: zone.saveReservation(); break;
            case PPAY: zone.savePayment(); break;
            case PLCONF: zone.savePlaces(); break;
            case FORMPAY: zone.openFormPayment(); break;
            case FORMRESPRIO: zone.openFormReservation(true); break;
            case FORMRESPLACES: zone.openFormPlaces(); break;
        }
    }
}

```