Devprogi Application

Développez votre propre logiciel

Table des matières

1 Introduction

1.1 Avertissement

Cet eBook n'a nullement la prétention de vous faire gagner des milles et des cents. Si c'était le cas, je ne pense pas que je m'efforcerai actuellement de faire un eBook et monter mon affaire sauf si j'avais l'intention de monter une escroquerie.
L'objectif de cet eBook est de vous montrer comment créer une application avec Devprogi application que vous pouvez télécharger gratuitement à l'adresse suivante :
www.devprogi.com
Bien qu'il y ait un message concernant la validité de l'application pendant 30 jours, sous réserve d'acheter la licence, il vous suffit de laisser votre adresse email pour obtenir le numéro de licence par mail
Ensuite, votre fortune provient de votre travail, et non d'une méthode miracle.
Tout dépendra de l'originalité de votre application et des méthodes de ventes. Je n'ai malheureusement aucunes méthodes concernant la vente.
En ce qui me concerne, j'ai longtemps cherché à savoir comment gagner un revenu. J'ai finalement opté pour le principe du don et la vente de cet eBook dans sa version intégrale.
Il existe donc une version lite de l'eBook pour se familiariser avec le produit et une version complète si vous êtes intéressé.

Dans cet eBook, vous apprendrez à utiliser Devprogi Application par un exemple concrêt. Je vous ferais développer une application de gestion en utilisant toutes les possibilités de Devprogi Application. Ensuite, libre à vous d'améliorer cet exemple et de vendre votre application ou de la donner à votre entourage.

1.2 Un bref historique

Je suis avant tout un developpeur Oracle, et je ne connaissais rien à java. Cependant, depuis longtemps je désirais être indépendant et éditer des logiciels aussi, Oracle se tournant vers JAVA, il était pour moi naturel de me former dessus. Je me suis donc amuser à réaliser quelques petits programmes seulement, constament écrire des lignes de codes pour le menu ou l'obligation de recompiler des que j'ajoutais un menu était devenu une contrainte, d'ou l'idée d'un template. Ayant travaillé sur Oracle Application, j'ai développé une interface dans laquelle je voulais juste me concentrer que sur l'aspect fonctionnel.

Je suis encore loin de mes objectifs mais je continue de travailler afin de les atteindre.
Ce qui est actuellement en place :

- ***gestion des menus*** :

La gestion du menu ne se fait pas par ligne de code mais par paramétrage.

- ***gestion des acces à la base de données*** :
  Quand vous développez un écran, vous n'avez pas à vous soucier de la gestion des acces à la base de données c'est à dire que les créations de données, les modifications ou les suppressions de données sont géré par l'interface.

- ***Gestion des messages & alertes***
  La gestion des messages est géré par l'appel d'une simple fonction à laquelle on passe le code d'un message crée dans l'interface.

- ***Gestion des champs date.***
  Lorsqu'un champ affiche une date, un bouton est associé à ce dernier et permet l'appel d'un calendrier.

- ***Gestion des mails***
  J'ai implémenté une gestion de mail afin de vous permettre de gérer l'envoi de mail.

- ***fonctions métiers***
  J'ai commencé à implémenter des fonctions finances. Elles ne sont pas nombreuses, mais au fil des versions, elles devraient être de plus en plus nombreuses. Je compte y intégrer des fonctions mathématiques (calcul intégral, statistique ...), plus de fonctions de finances, des fonctions scientifiques

- ***Internationalisation***
  En java, l'internationnalisation peut être aisé avec les fichiers ressources. Une fonction est mis à disposition pour mettre en place une application multi-langage.

- ***clé de vérrouillage de l'application***.
  Lorsque vous avez terminé votre application, vous avez la possibilité de mettre en place une clé afin de verrouiller l'application au bout d'un certain.

D'autres fonctionnalités sont ou seront développées et je vous conseille de venir régulièrement sur le site pour consulter les divers évolutions. Je les développe au fur et à mesure que je développe des applications. Par exemple, lorsque j'ai développé Devprogi Mailing, j'ai intégré les fonctions d'envoi de mail. Je développe actuellement une application de gestion de compte aussi ai-je eu besoin de nouvelles fonctionnalités que j'ai aussitôt intégré à Devprogi Application.

Bon résumons avant de rentrer dans le vif du sujet, les longues nuits à développer, avec la sueur au front lorsqu'un soucis technique ne peut être résolu trivialement, les nuits ou les meilleurs amis sont le café et les cloppes (tout au moins pour ceux qui fument) je me les garde, je ferai don de mes poumons aux sociétés d'autoroute pour en récupérer le goudron et quant à vous, vous faites le plus simple... enfin c'est l'objectif.

2 Installation et configuration

2.1 Installation

Une fois que vous avez réussi à télécharger le fichier excutable, vous lancez l'exécutable comme n'importe quel autre fichier d'installation. Je vous conseille dans un premier temps de garder le chemin d'installation proposé. Le lancement peut se faire automatiquement.
Lors du lancement, un répertoire database est généré, ainsi que 2 fichiers. « DevProgi_fr.properties » contenant les libellés de certains menus et « devprogi.ini », contenant les différents chemins. Il faut noter que ce dernier contient divers paramètrage de votre application. Ainsi, c'est lui qui détermine si vous voulez une barre d'outils ou si vous voulez afficher la fenêtre d'identification.

2.2 Caractéristiques actuelles

2.2.1 Base de données internes

Bien que les premières versions furent faites avec Oracle ou Mysql, j'ai opté pour une base de données est une base de données interne « One$db4 ». L'avantage est qu'il n'est pas nécessaires de lancer des scripts de création de tables systèmes. L'objectif finale est de permettre de pouvoir utiliser différentes base de données au choix ainsi que la possibilité d'utiliser plusieurs bases de données (par exemple installer les tables systeme sur une base et les tables de l'application sur une autre).

2.2.2 téléchargement d'applications

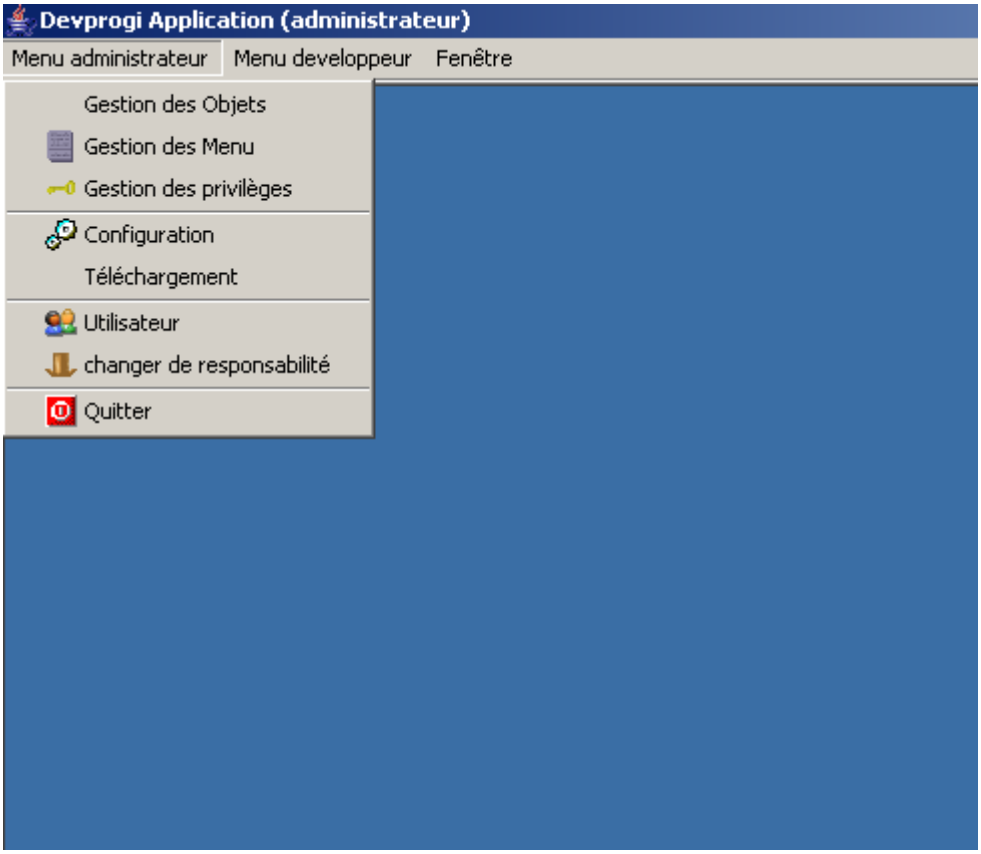
L'objectif finale est de permettre au client de pouvoir télécharger et installer automatiquement des modules supplémentaires et ainsi de pouvoir compléter son application au fur et à mesure de ses besoins.

3 Utiliser Devprogi Application

Maintenant nous allons passer en revue les différents menus.

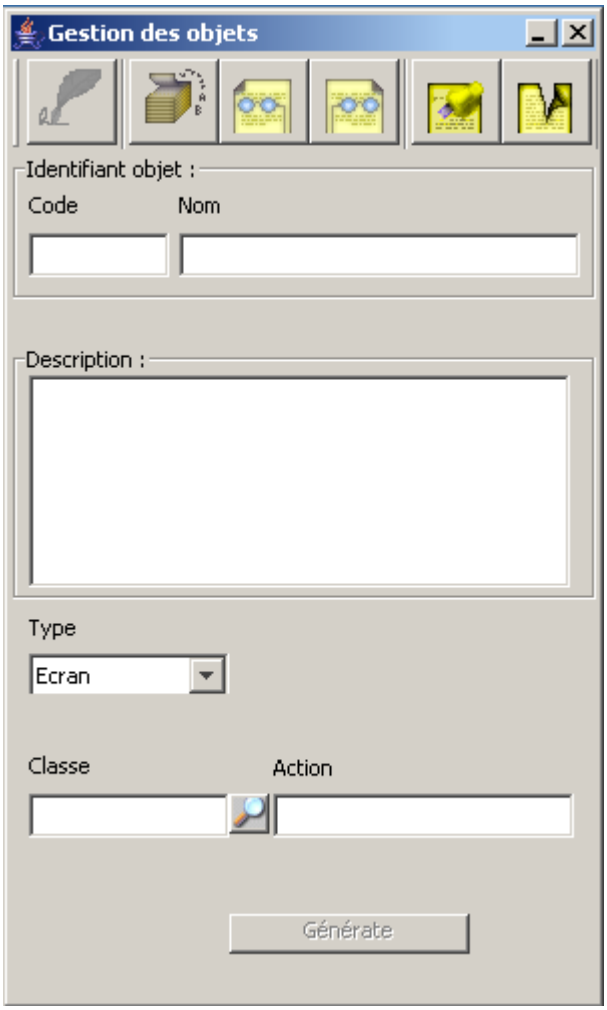
3.1 Menu administrateur

Le menu administrateur permet la gestion de l'application. On y gère l'acces aux différents objets et menus aux utilisateurs. C'est dans ce menu que l'on gère les différents privilèges accordés aux utilisateurs pour l'accès au différents écrans.



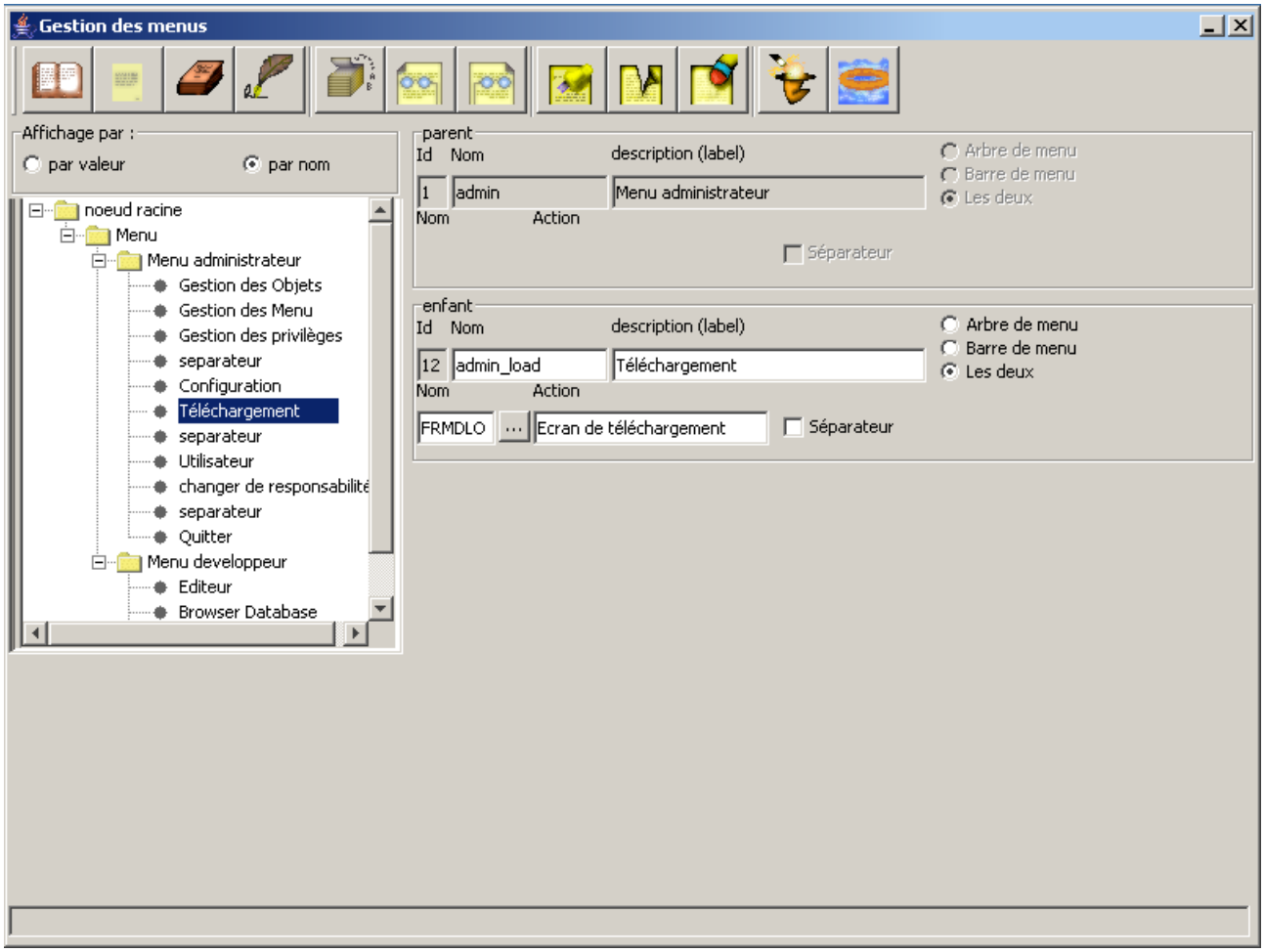
3.1.1 Gestion des Objets

Lorsque du coté développeur, on a une classe, on l'associe à un objet (métadonnée). L'objectif est par ailleurs de pouvoir réutiliser une même classe suivant différents paramètres. Le passage de paramètres à travers un objet n'est pas encore implémenté. Lorsque dans une classe on veut faire appel à une seconde classe, on fait appelle à l'objet attaché à cette dernière.



3.1.2 Gestion des menus

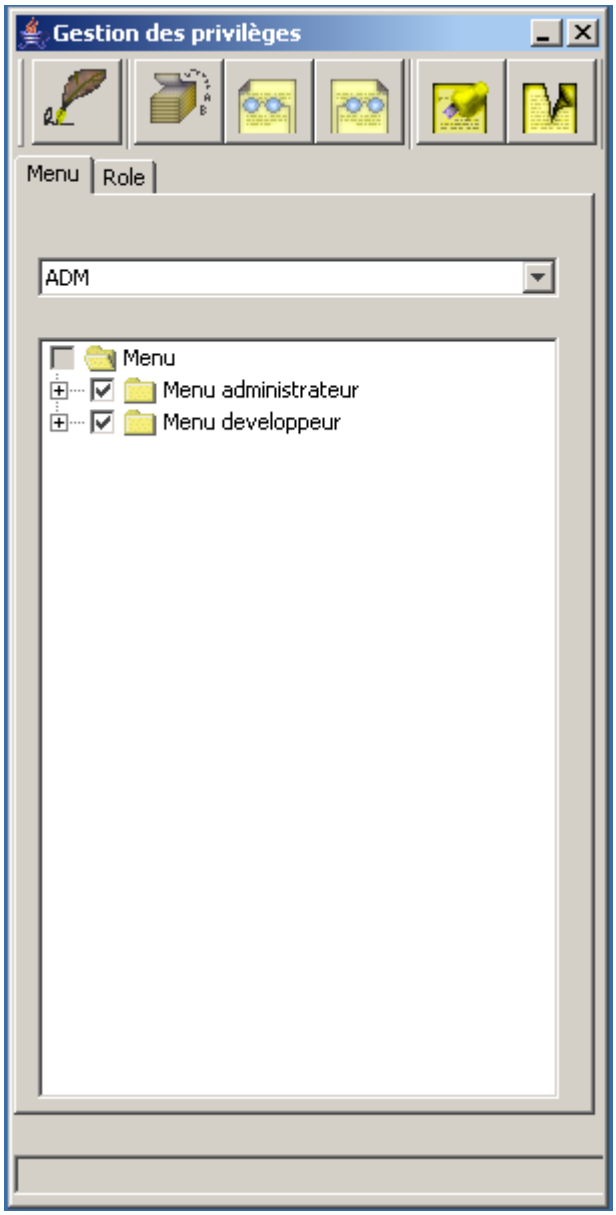
Cet écran permet la création de menu et de sous-menu. On peut choisir d'afficher le menu soit dans la barre de menu, soit l'arbre soit les deux.



Cet écran devrait être dans un proche avenir être totalement revu et corriger.

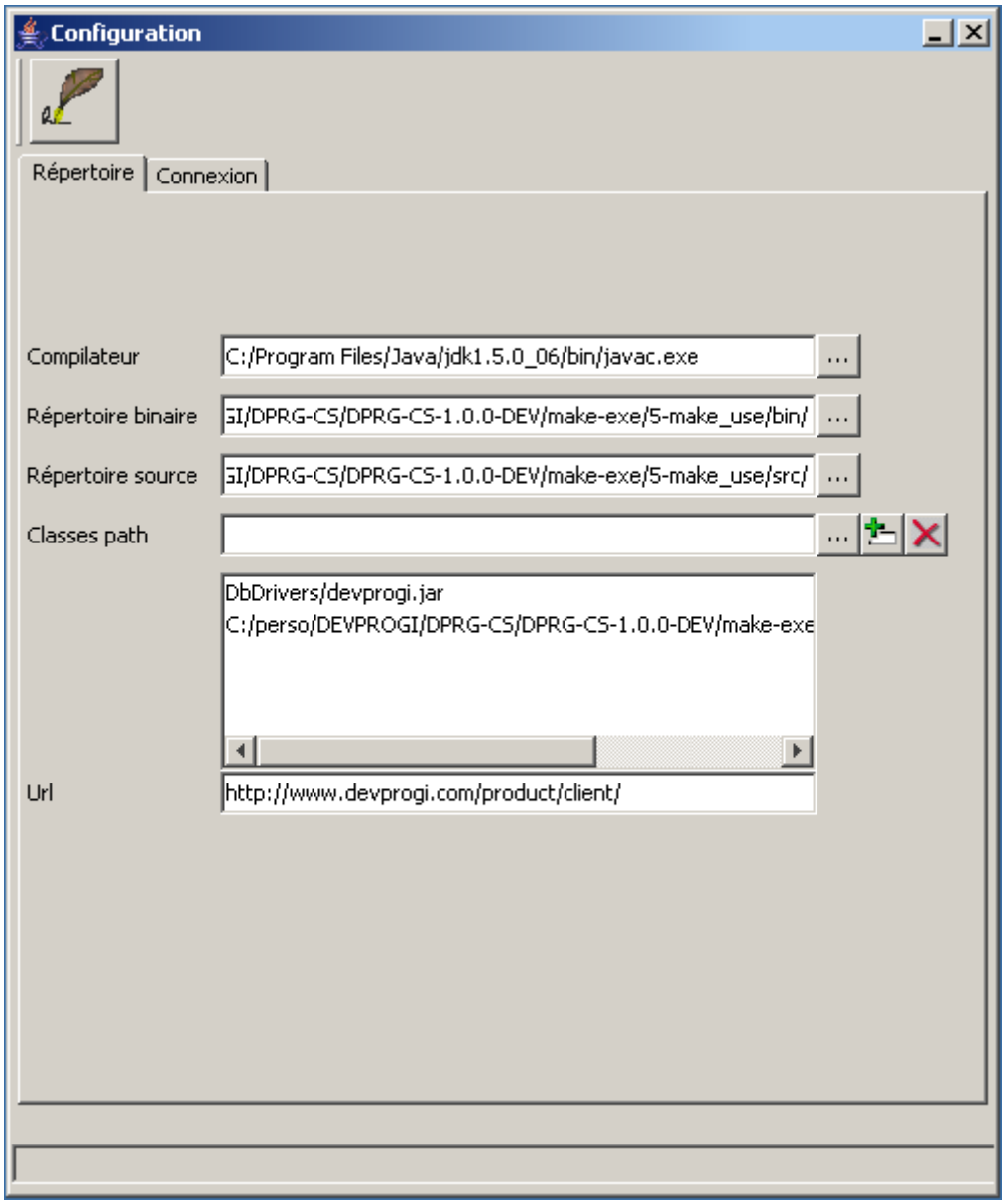
3.1.3 Gestion des privilèges

Cet écran permet quant lui d'accorder l'accessibilité du menu à un profile utilisateur.



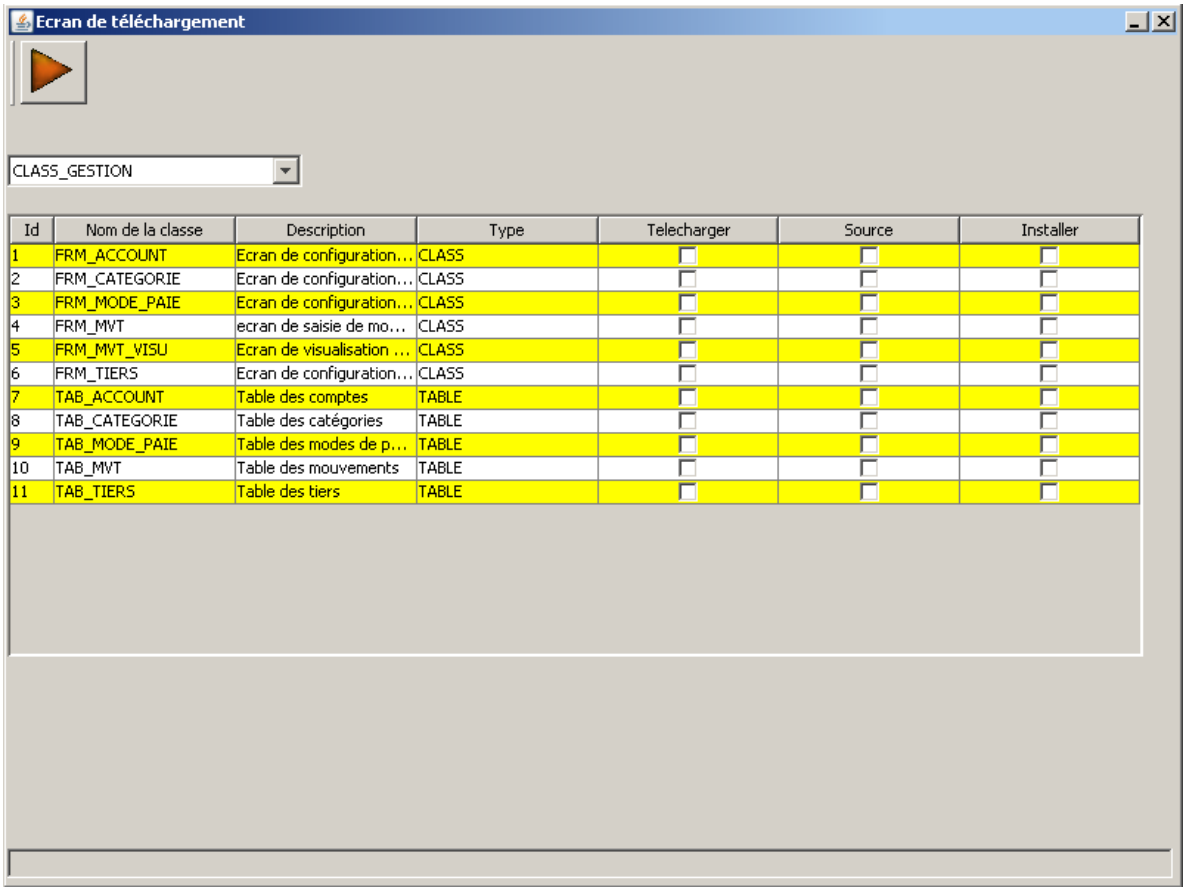
3.1.4 Configuration

Cet écran permet surtout de positionner les différentes variables d'environnement.



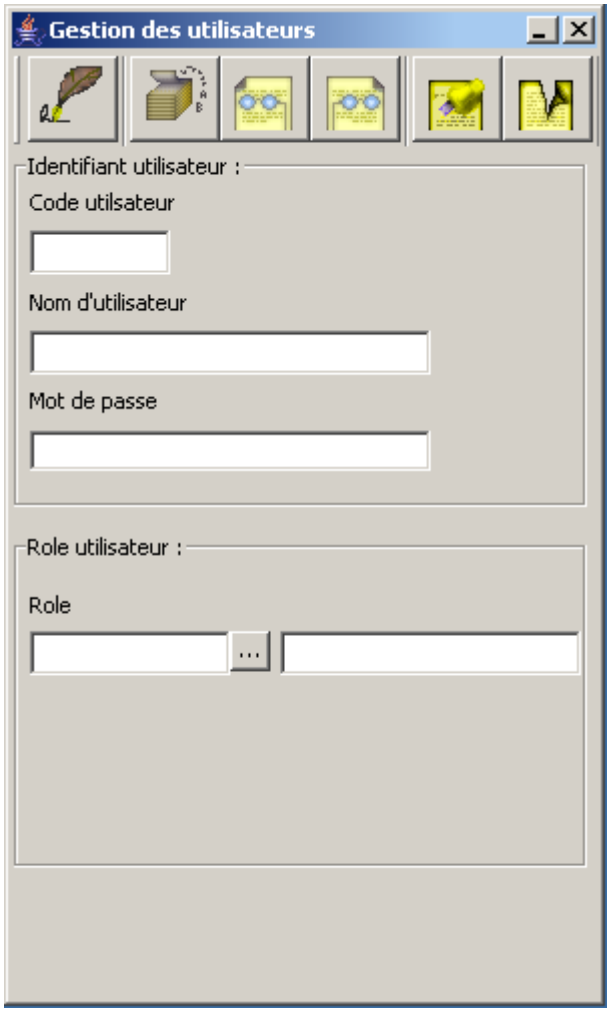
3.1.5 Téléchargement

L'ecran de téléchargement permet de télécharger du site Devprogi des classes supplémentaires.



3.1.6 Utilisateurs

Ce dernier permet la création d'utilisateur et de leur attribuer un profile.

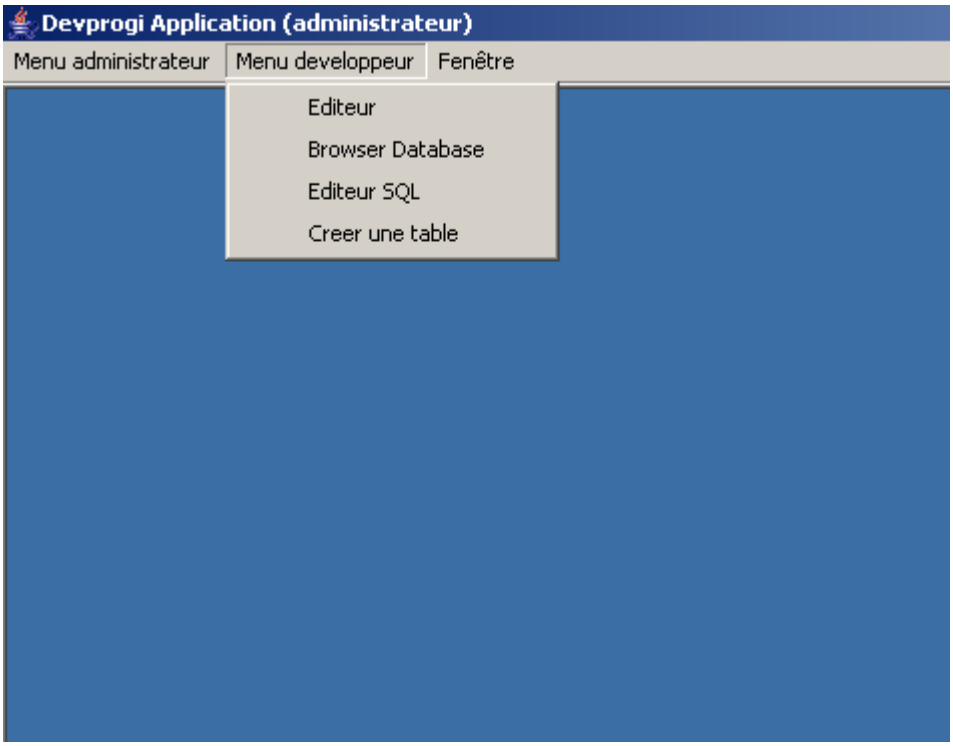


3.1.7 Changer de responsabilités

Ce menu permet de changer de connection utilisateur autrement dit on peut si on veut, passer d'un utilisateur avec un rôle d'administrateur ou développeur à un rôle d'utilisateur simple.

3.2 Menu développeurs

Le menu développeur contient pratiquement tous les outils permettant de développer des écrans, des reports ou des graphiques (même si ces deux dernières fonctionnalités ne sont pas encore implémentées).



3.2.1 Editeur

L'éditeur de classe permet de développer une classe en prenant en compte l'environnement de l'application. Dés lors que l'on crée une nouvelle classe, plusieurs possibilités de template sont proposées. On peut créer soit un écran, soit un état, soit un graphique. Un nouvel onglet devrait être rajouté pour des traitements, des fonctions ou méthodes. Actuellement, seul l'onglet « Ecran » est le plus finalisé. On propose divers templates comme créer un écran simple, un écran basé mode formulaire ou un écran basé en mode tableau.

3.2.2 Browser Database

Cet outil donne la possibilité d'acceder en lecture aux différents de la base de données interne.

3.2.3 Editeur SQL

L'editeur SQL permet comme son nom l'indique d'effectuer toutes les requetes SQL sur la base de données interne.

3.2.4 Créer une table

Ce dernier permet de lancer un assistant de création de table. Cet écran nécessite encore de nombreuses évolutions sachant qu'il ne permet que la création et non l'ouverture et la modification d'une table.

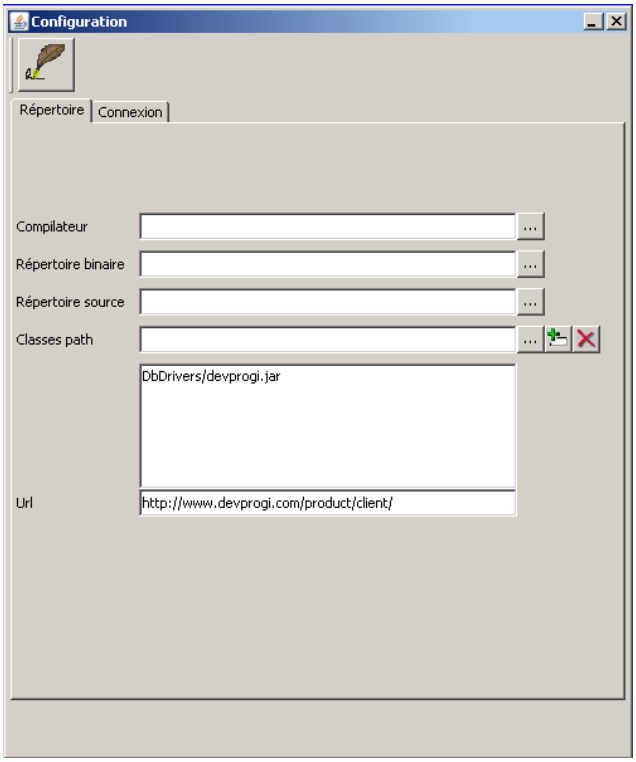
4 Developpement avec Devprogi Application

4.1 Lancement de l'application & connexion

Après le lancement de l'application, vous vous connectez en tant que administrateur.
Nom d'utilisateur : administrateur **mot de passe** : test

4.2 Configuration de Devprogi Application

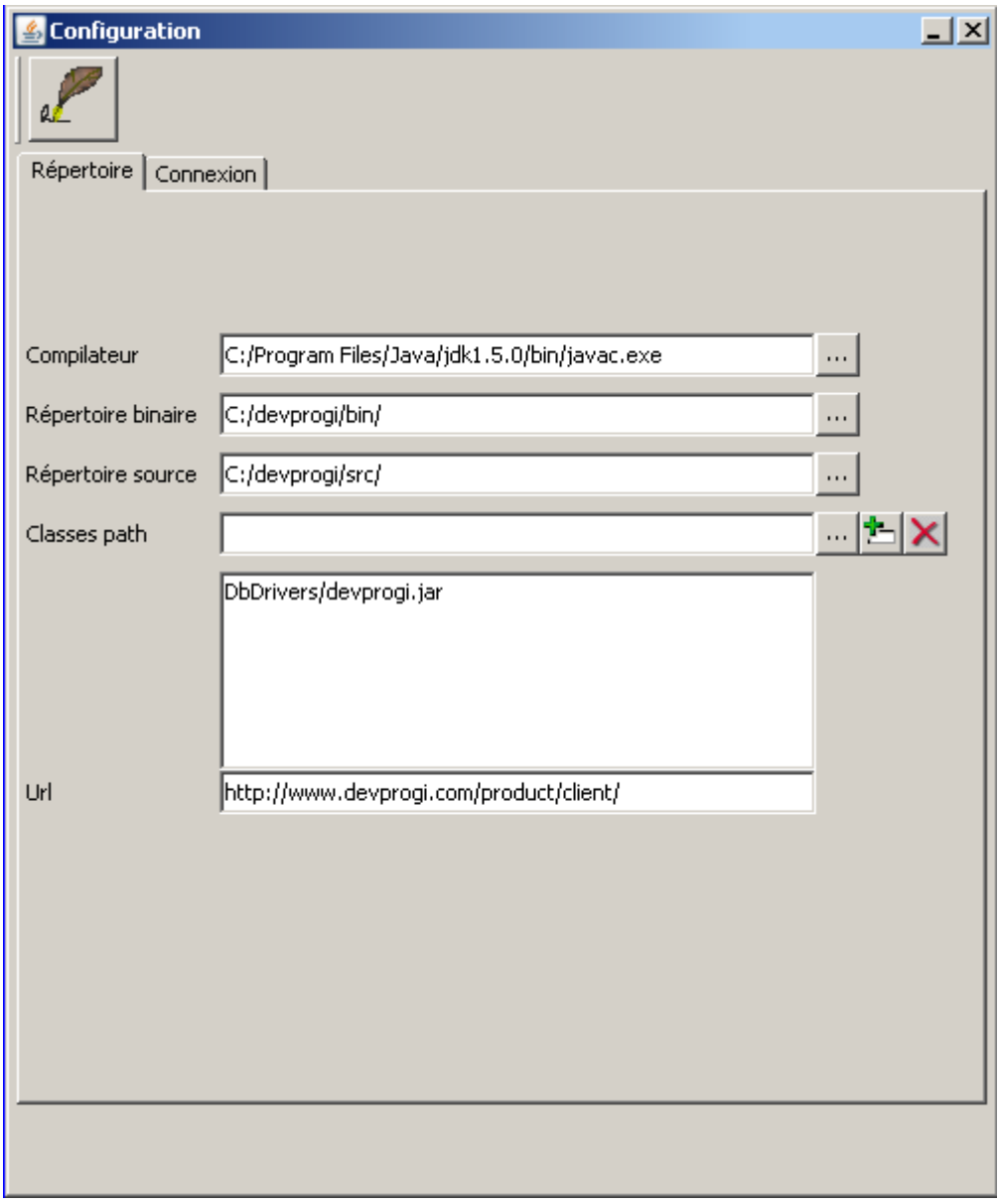
Avant toute chose, il est nécessaire de configurer l'application.
Pour ce faire, vous devez lancer l'écran de configuration.
Menu administrateur / Configuration



Le compilateur est celui définit dans le java home directory (voir figure précédente).

Le répertoire binaire est le répertoire dans lequel sera stocké les classes.

Le répertoire source est le répertoire dans lequel seront sauvé les sources de l'application. Ces deux répertoires sont à créer manuellement.



5 Réalisation d'une application

Nous allons maintenant construire une application afin que vous puissiez découvrir les différentes possibilités qu'offre Devprogi Application.

5.1 Objectif

Nous allons réaliser une application de gestion de compte (application assez simple mais permettant de démontrer différentes possibilités de Devprogi Application).

5.2 Création des tables

Les tables que nous allons utiliser seront

- La tables des comptes (TAB_ACCOUNT) permettant de stocker le ou les comptes bancaires.
- La table des mouvements (TAB_MVT) permettant d'enregistrer les différentes opérations sur le compte bancaire.
- La table des catégories (TAB_CAT)
- la table des tiers (TAB_TIERS).
- La table des mode de paiement (TAB_MP)

Pour la création des tables, nous allons dans le menu

menu developpeur / créer une table

Cet outils n'est toujours pas finalisé. Il est clair que les prochaines versions devront intégrer un outils plus complet. Actuellement, j'utilise plus un script SQL que j'éxécute dans la fénêtre d'éditeur SQL. Mais à l'avenir, je compte réaliser un éditeur d'objet plus performant pour modifier, supprimer ou créer des objets tel que tables, vues ou triggers.

Enfin nous allons créer la première table, la table des mouvements. Par soucis de simplification, nous aurons comme colonne

1.	l'ID de la ligne	MVT_ID	int PK
2.	le montant du mouvement	MVT_AMOUNT	Number(16,2)
3.	la date de transaction	MVT_DATE	Date
4.	le mode de paiement	MVT_MP_CODE	Varchar(10)
5.	le tiers	MVT_TIERS_CODE	Varchar(10)
6.	le compte	MVT_ACCOUNT	Varchar(10)

Pour la table des tiers (toujours simplifié)

1. l'identifiant du tier	TIERS_ID	int PK
2. le code tiers	TIERS_CODE	Varchar(10)
3. le nom du tiers	TIERS_NAME	Varchar(100)
4. un commentaire	TIERS_COMMENT	Varcha(300)

Pour la table des comptes

1. l'ID	ACC_ID	int
2. le code	ACC_CODE	varchar(10)
3. le nom de la banque	ACC_BANK	varchar(100)
4. le code compte	ACC_NUMBER	varchar(11)
5. le code banque	ACC_CD_BANK	varchar(5)
6. le code guichet	ACC_CD_GUIC	varchar(5)
7. la clé RIB	ACC_KEY	varchar(2)

Pour la table des modes de paiement

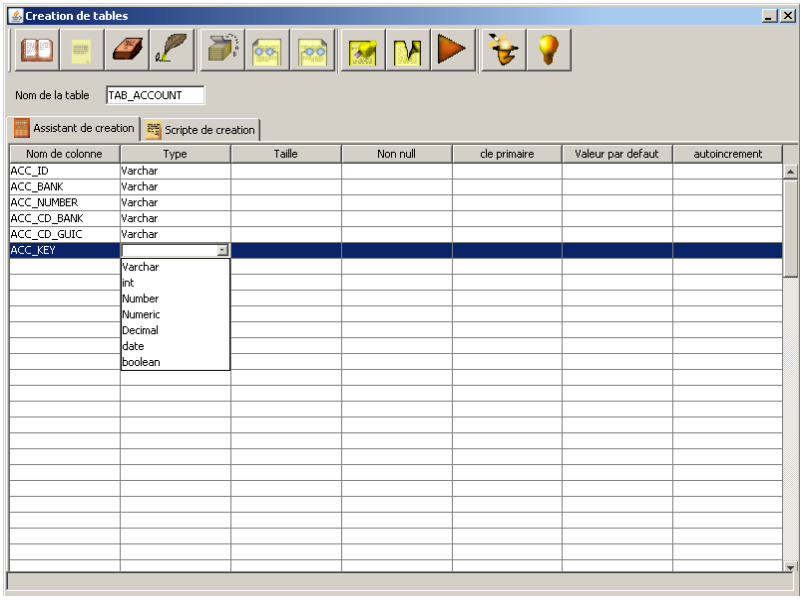
1. l'identifiant	MP_ID	int
2. le code mode de paiement	MP_CODE	varchar(10)
3. le mode de paiement	MP_NAME	varchar(40)
4. la description	MP_DESC	varchar(100)

Et enfin la table des catégories. A noter que pour des raisons de simplicité, nous ne prenons pas en compte des sous-catégories.

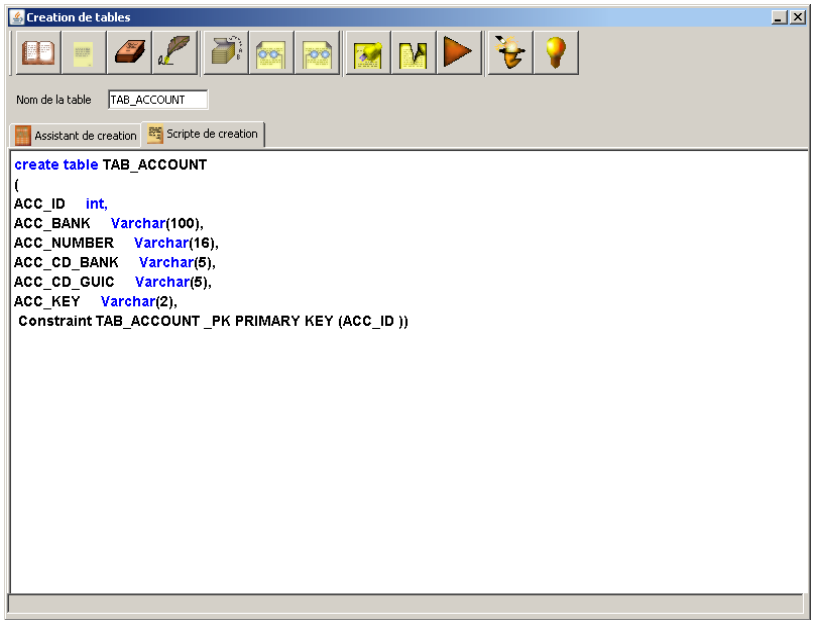
5. l'identifiant	CAT_ID	int
6. le code mode de paiement	CAT_CODE	varchar(10)
7. le mode de paiement	CAT_NAME	varchar(100)
8. la description	CAT_DESC	varchar(200)

Maintenant, si vous n'est pas trop branché SQL, Vous avez la possibilité de créer les tables via l'écran prévu à cette effet, ou sinon, dans le cas contraire, vous rendre dans l'éditeur SQL.

Dans l'écran de création des tables, vous saisissez le nom de la table et vous saisissez ensuite les colonnes. Avant de lancer la création, il est fortement recommendé de sauvegarder le script de création de la table.



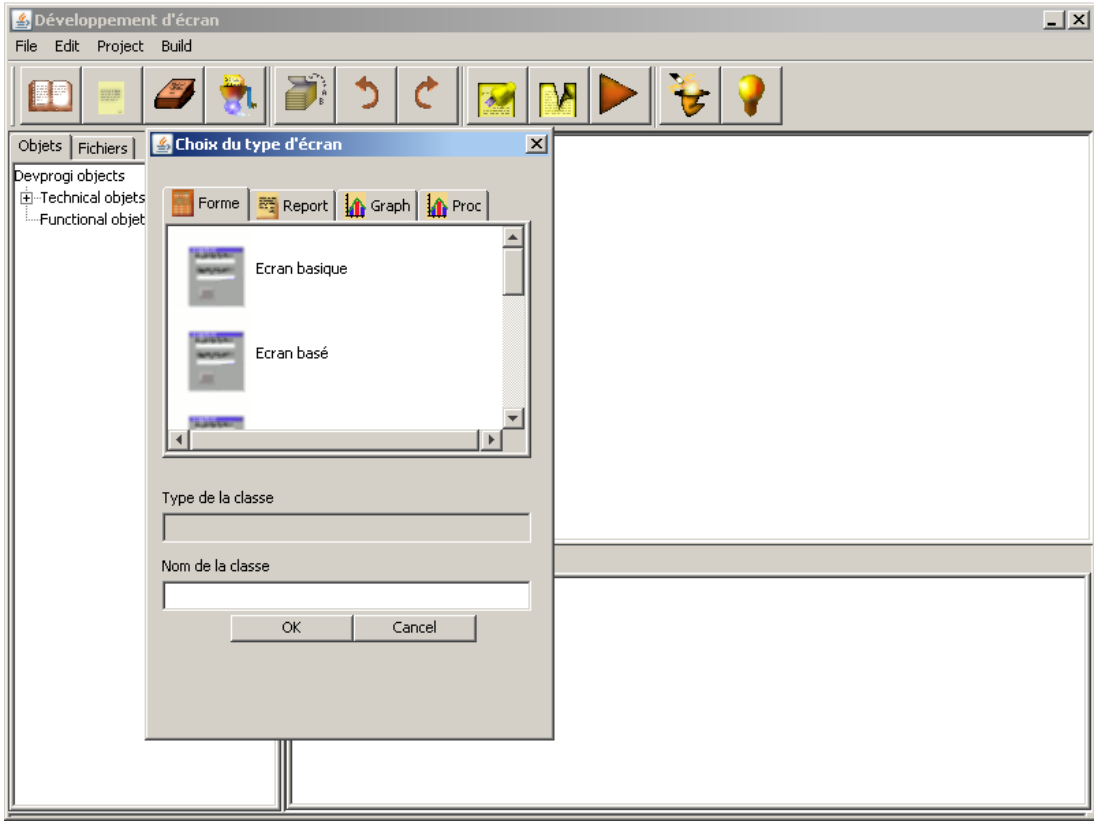
Comme vous pouvez le constater, il n'y a pas de prise en compte de l'option AUTOINCREMENT. C'est une des nombreux points à améliorer. Je vous conseille de saugarder le script et éventuellement le corriger.



5.3 Création des écrans

Maintenant, nous allons passer au choses sérieuses. Nous allons maintenant nous familiariser avec l'éditeur de classe.

Menu développeur / Editeur



Une fois lancé, vous appuyer sur le bouton « nouveau »

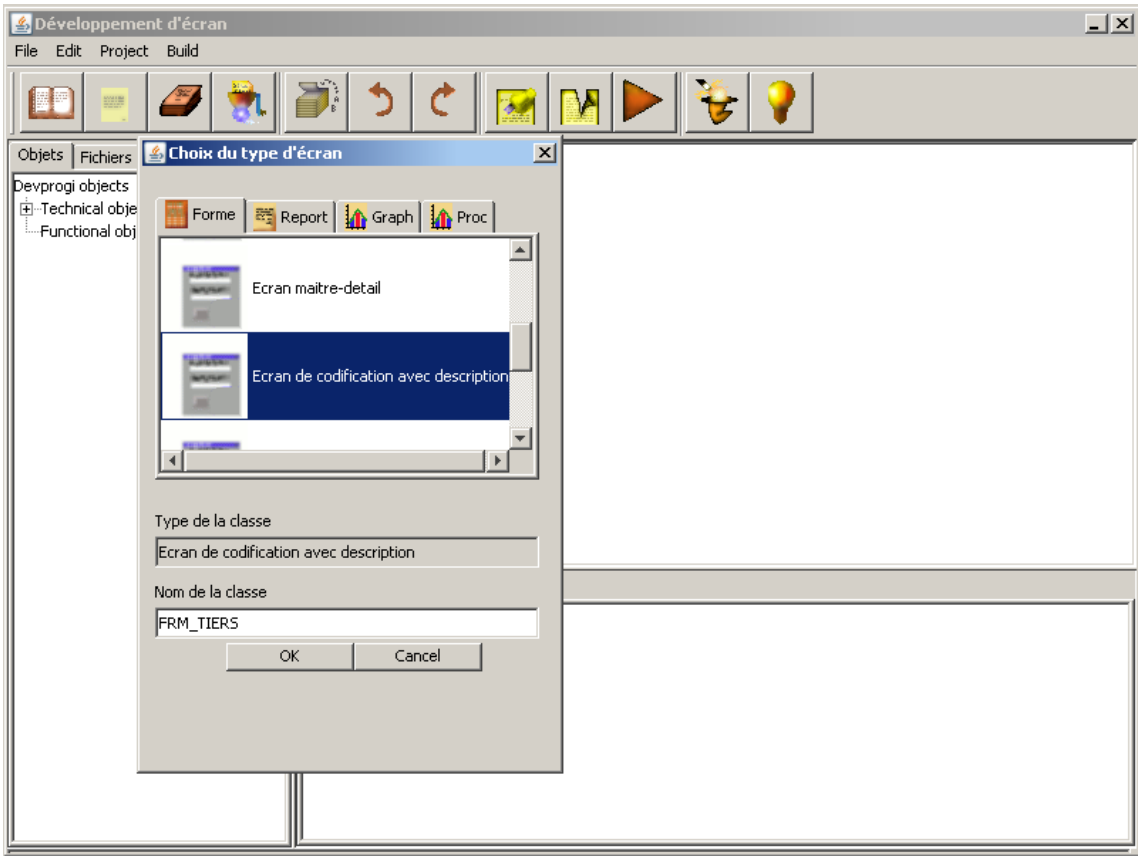
Vous avez à disposition, différents templates. Certains étant finalisés, d'autre non encore finalisé. Je mettrais par la suite de nouveau template au fur et à mesure que l'application évoluera.

Pour commencer, nous allons faire ensemble l'écran des tiers puis les prochains écrans, à l'exception de l'écran des mouvements.

5.3.1 Ecran des tiers

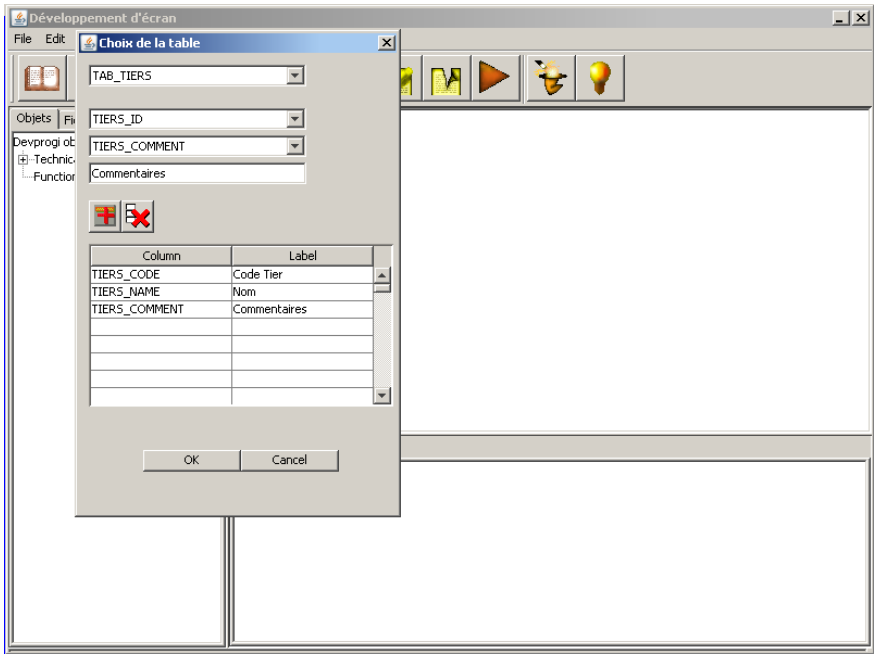
5.3.1.1 Création de la classe FRM_TIERS

Le nom donné à la classe de l'écran des tiers est FRM_TIERS. Nous faisons donc nouveau et nous choisissons comme template « Ecran de codification avec description ».

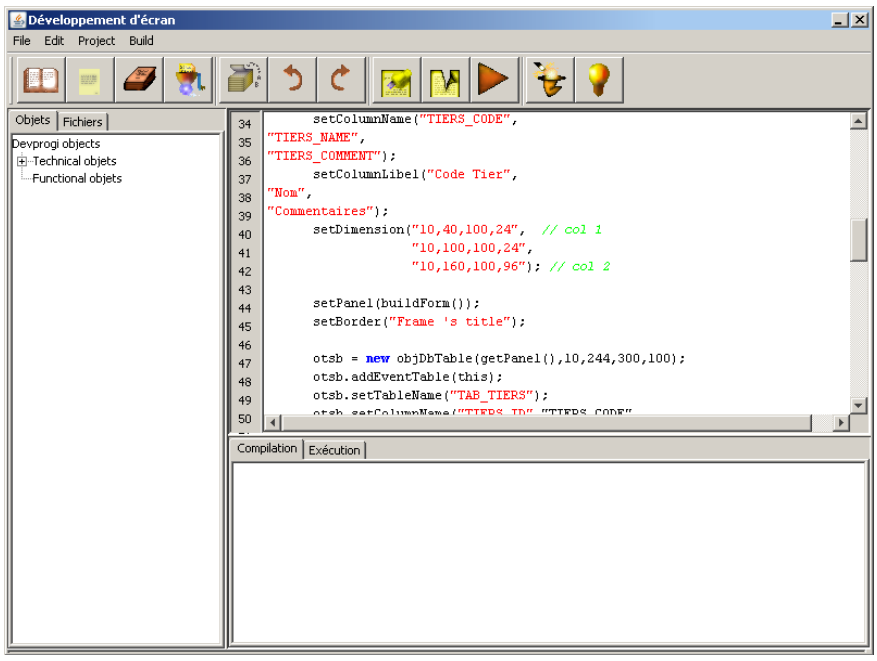


Nous y saisissons aussi le nom de la classe.

Une fois que nous avons fait OK, une nouvelle fenêtre s'ouvre dans laquelle vous allez pouvoir choisir la table sur laquelle l'écran s'appuiera et les colonnes qui seront les futurs champs de l'écran. Je reconnais que c'est aussi un écran que je dois améliorer. Dans cette version, les champs sont qui seront générés seront de simples champs texte.



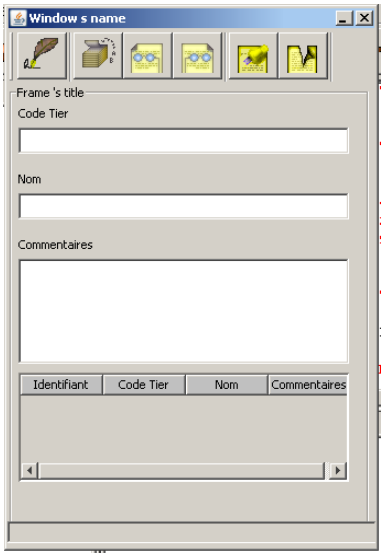
Une fois que vous avez cliqué sur OK, le code est généré. Il ne vous reste plus qu'à le compiler en cliquant sur le bouton suivant :



Vous pouvez maintenant executer votre premier programme sous Devprogi Application n

cliquant sur le bouton

Et si tout se passe correctement (priez Saint Devprogi, patron de l'application), vous devriez voir l'écran suivant :

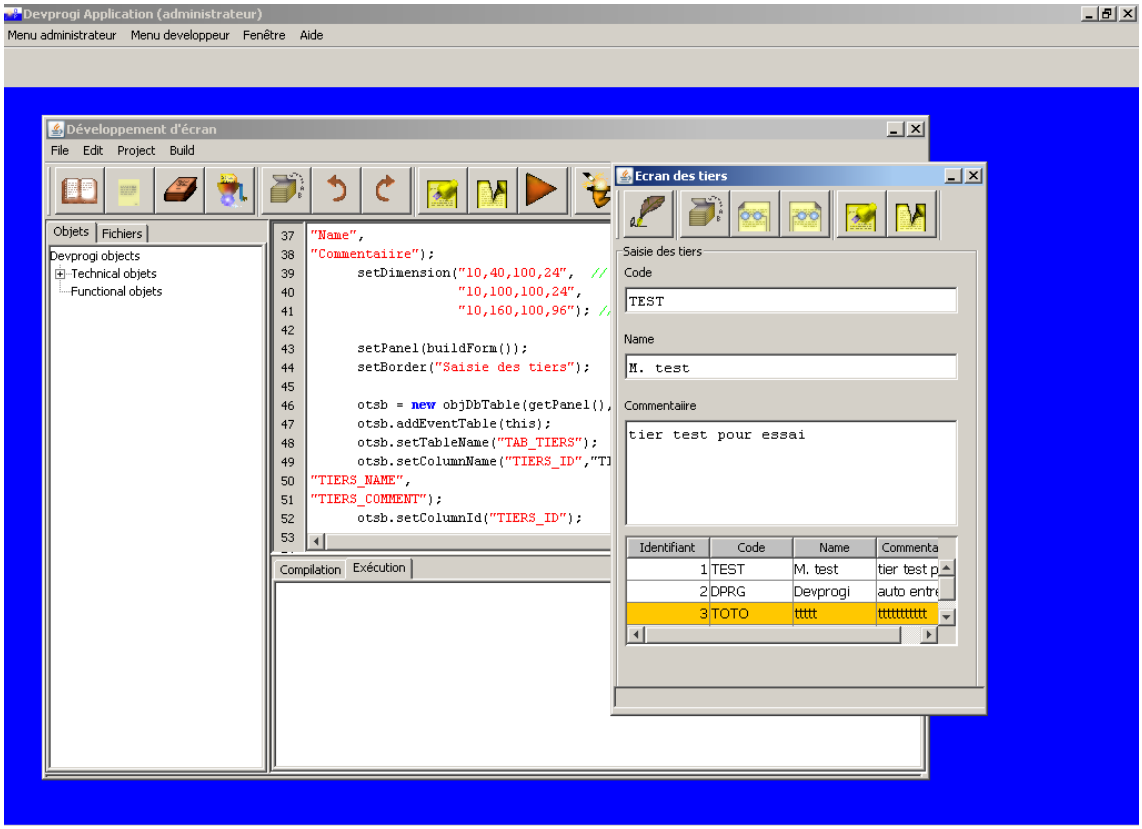


Vous pouvez remarquer que le titre de la fenêtre et le titre de la frame sont des titres génériques. Vous devez aller dans le code afin de les modifier

```
super("Ecran des tiers");
```

et

```
setBorder("Saisie des tiers");
```



Maintenant, voyons un peu quelques caractéristiques et le comportement de l'écran. Commençons par les icônes

 Sauvegarder les données saisies

 Afficher les données (premier enregistrement)

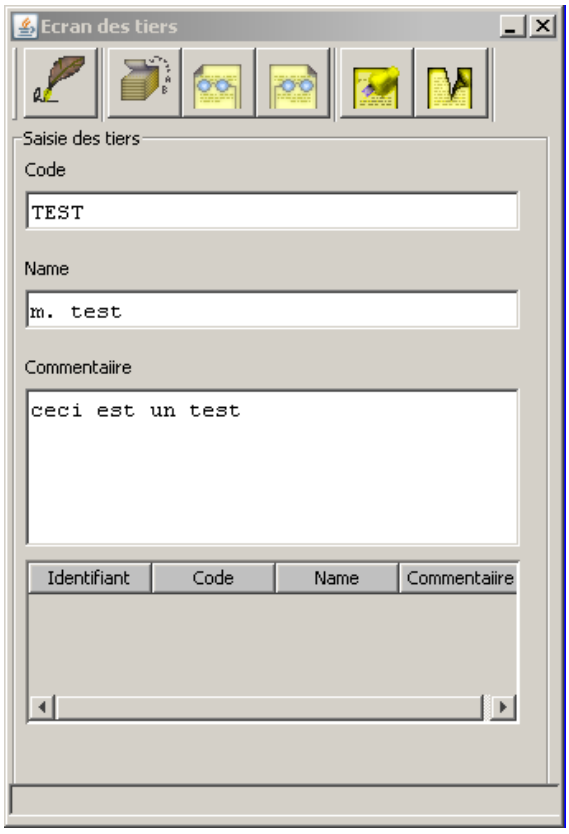
 Enregistrements suivants

 Enregistrements précédents

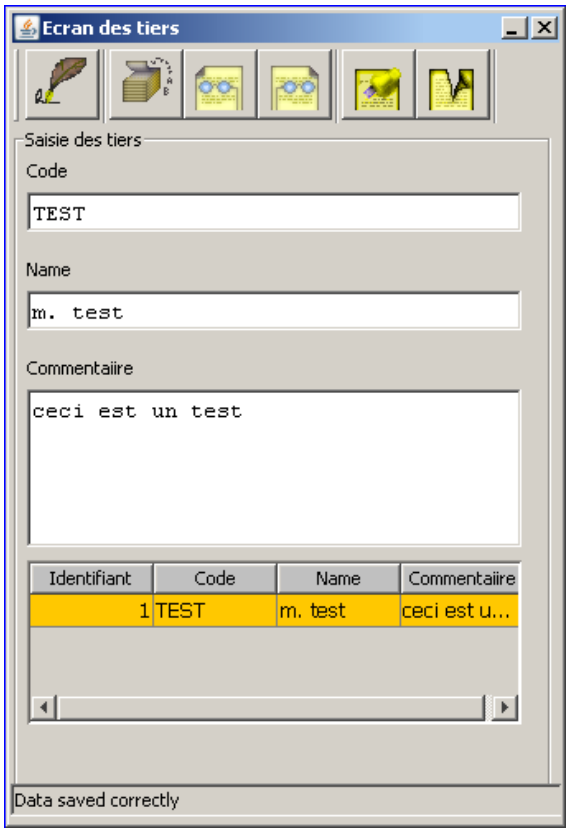
 Nouvelle saisie

 Supprimer un enregistrement

Maintenant, saisissons un tiers. Losrqu'on sauvegarde, une ligne apparaît dans le tableau.



Avant sauvegarde



Après sauvegarde

Par la suite, si vous voulez modifier un enregistrement, il vous suffit de cliquer sur un des enregistrements du tableau.

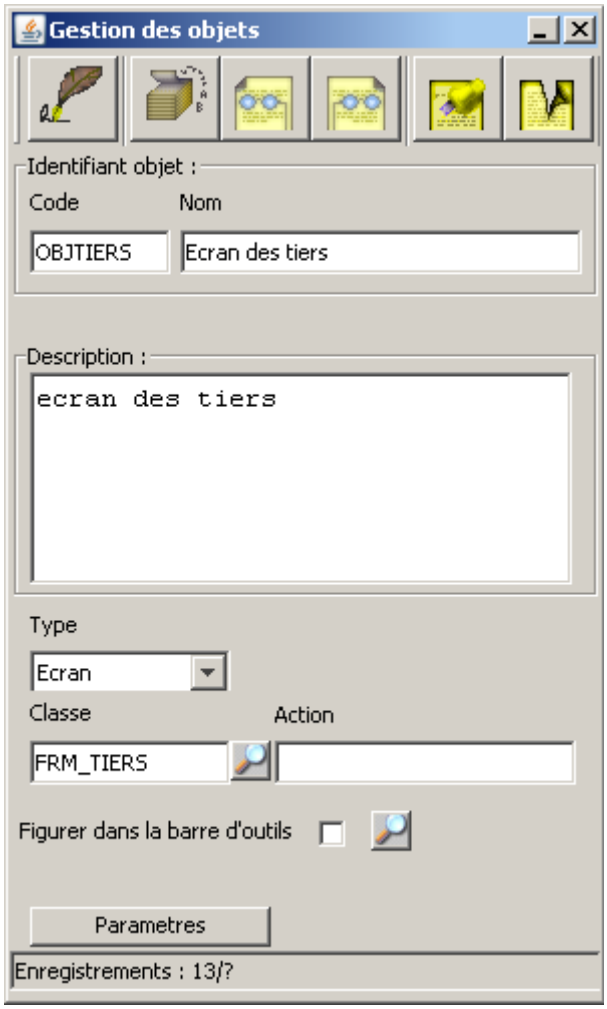
5.3.1.2 Création de l'objet associé à la classe FRM_TIERS

Nous allons maintenant créer l'objet permettant par la suite l'appel de la classe FRM_TIERS par le menu ou une autre classe.

On ouvre l'ecran

Menu administrateur / Gestion des objets

On crée alors l'enregistrement de l'objet

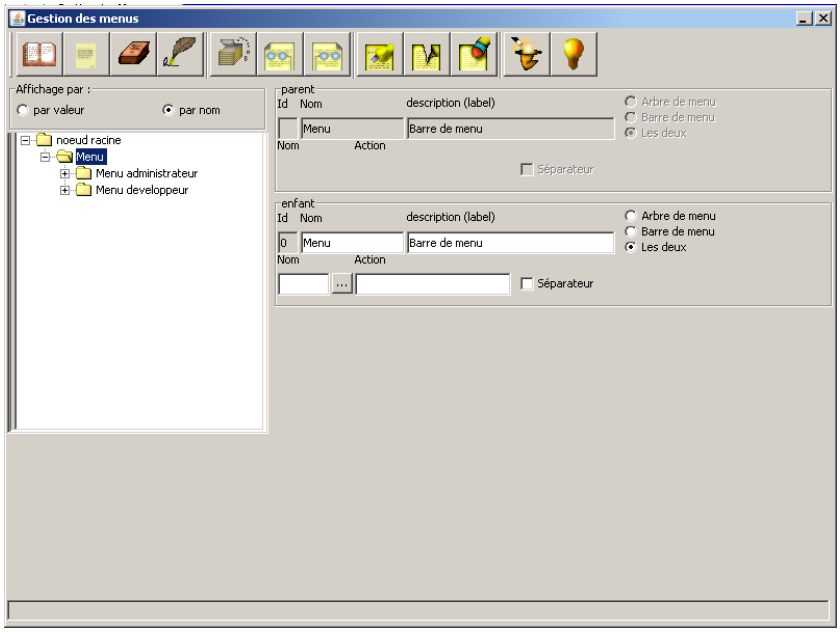


Cet écran aussi est amené à être complètement modifié dans le futur.

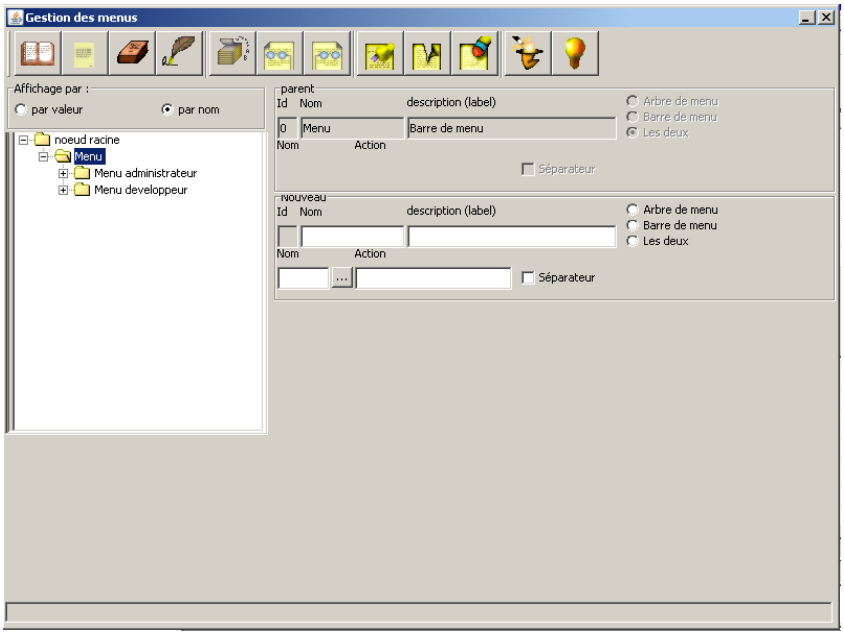
5.3.1.3 Creation du menu Tiers

Nous allons maintenant ouvrir l'écran de gestion des menus :

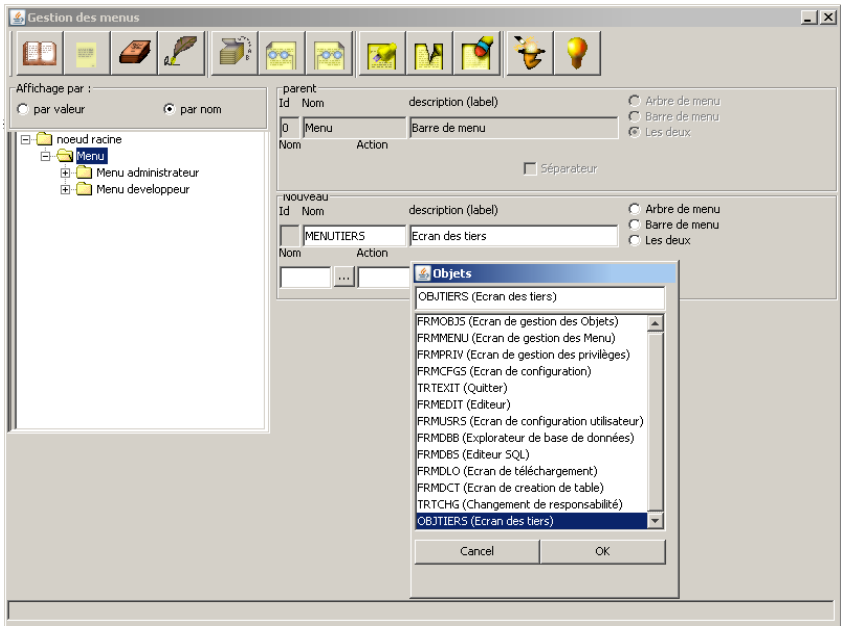
Menu administrateur / Gestion des menus

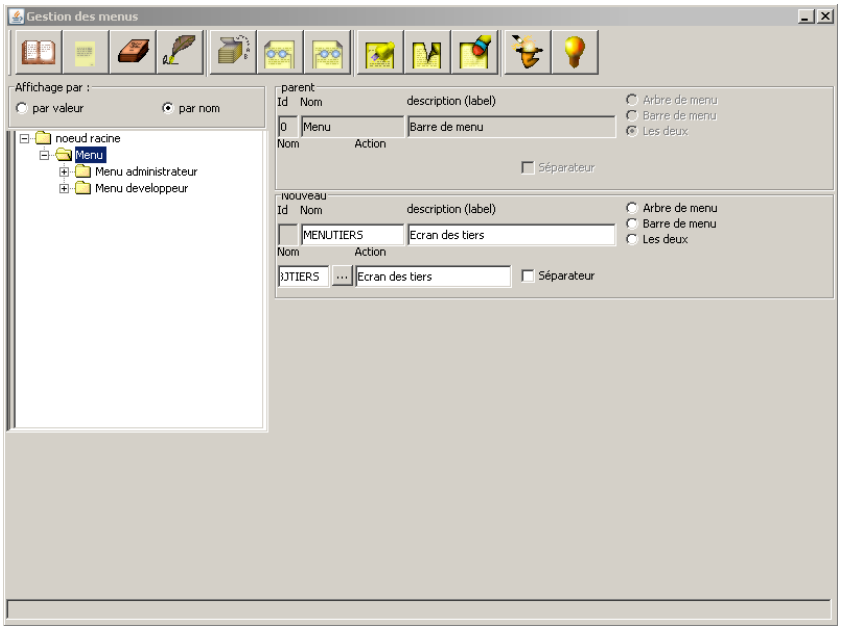


Positionner vous sur le menu « père » et appuyer sur « nouvel enregistrement » 

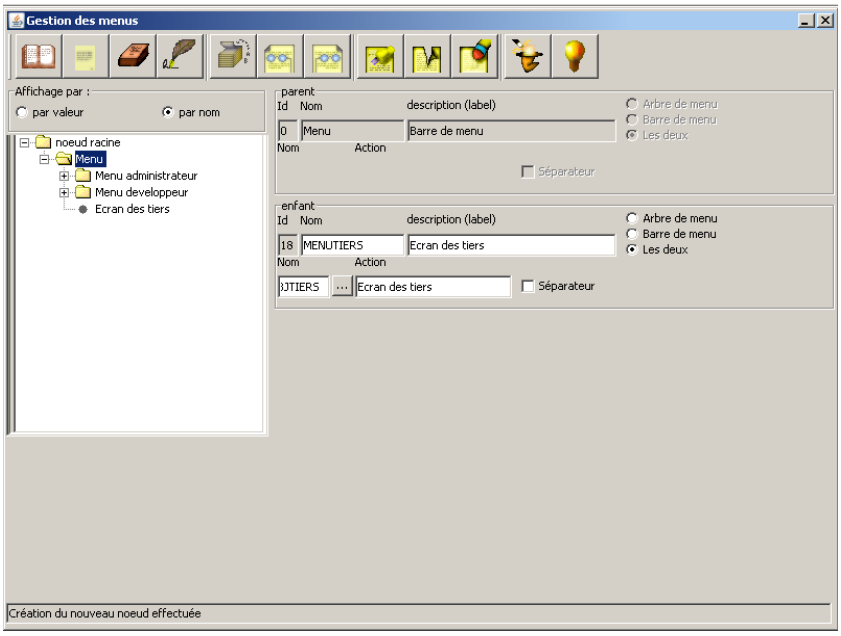


et enfin saisissez le nouveau menu. L'action représente l'objet qui permettra l'éxécution de la classe, en d'autres termes, en ce qui nous concerne, l'objet créé précédemment.

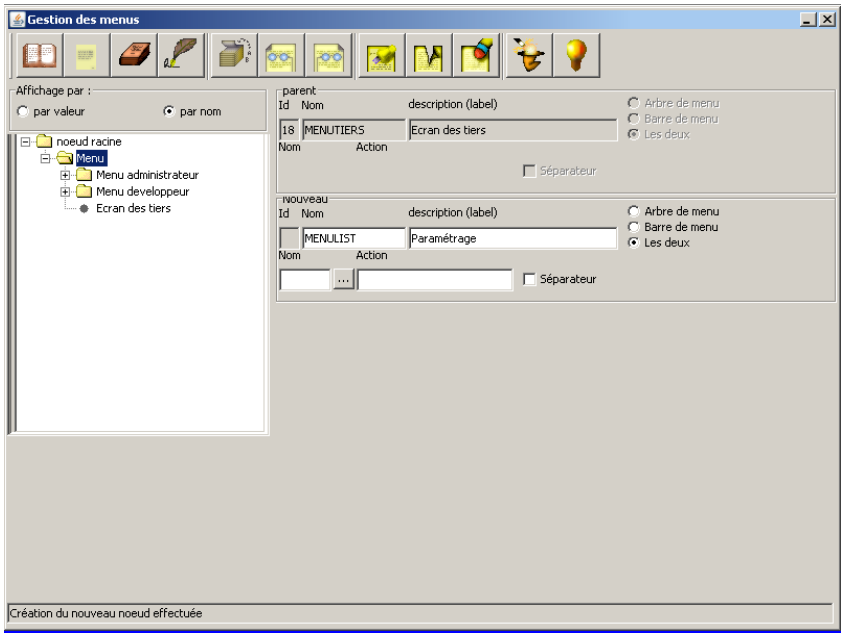


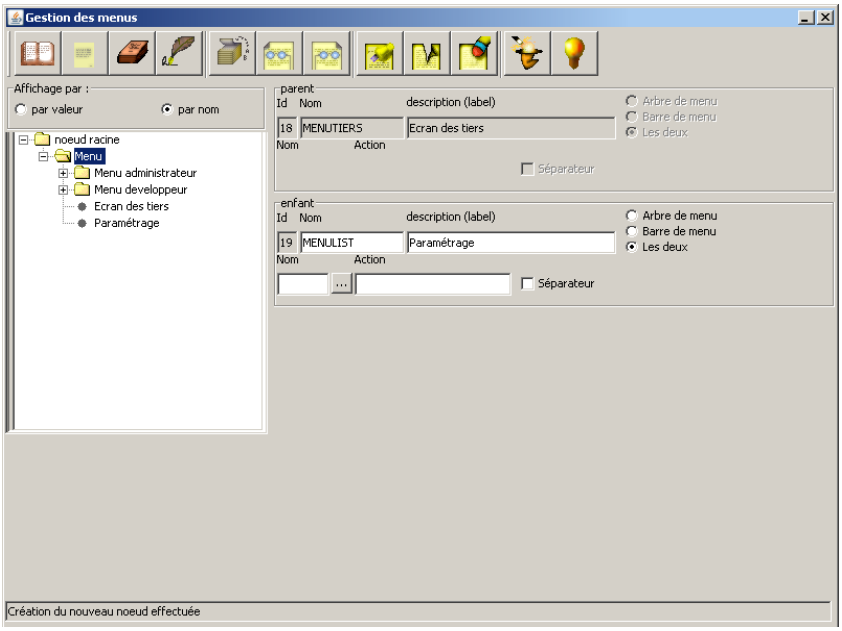


Une fois saisi, on clique sur le bouton « sauvegarder » 

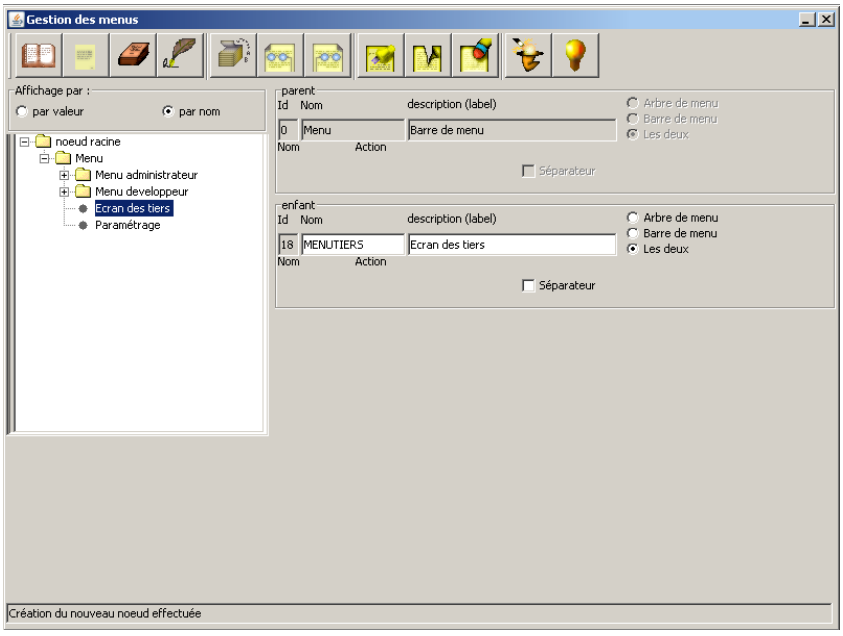


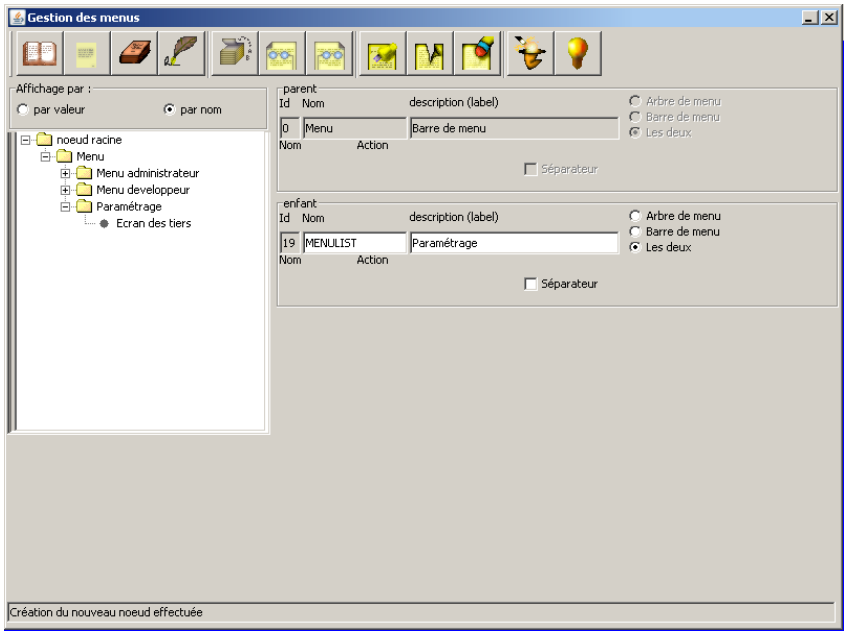
Nous avons donc dans la barre de menu, un menu permettant d'ouvrir l' écran des tiers mais ce n'est encore guère satisfaisant. On aurait plutôt envie de pouvoir le lancer d'un sous-menu. Alors, soyons fou, créons un menu dans lequel nous introduirons le menu précédemment créé. Nous commençons par sélectionner le menu parent de ce futur noeud à savoir, le noeud « menu » en cliquant sur « nouveau ».





Remarquez que pour un noeud parent, nous ne saisissons rien au niveau de l'action.Enfin, nous sélectionnons le premier noeud correspondant au menu « Ecran des tiers » et nous le glissons sur le dernier noeud créé (drag & drop).

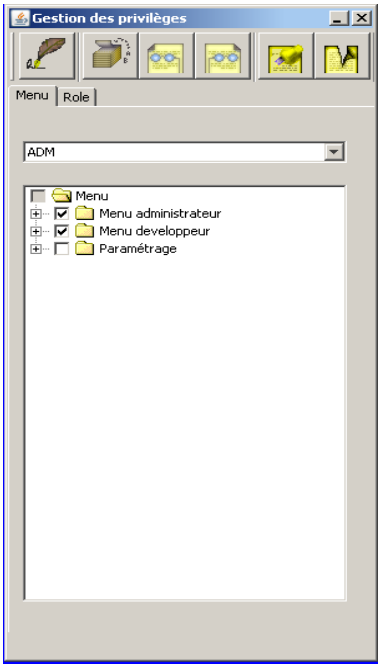




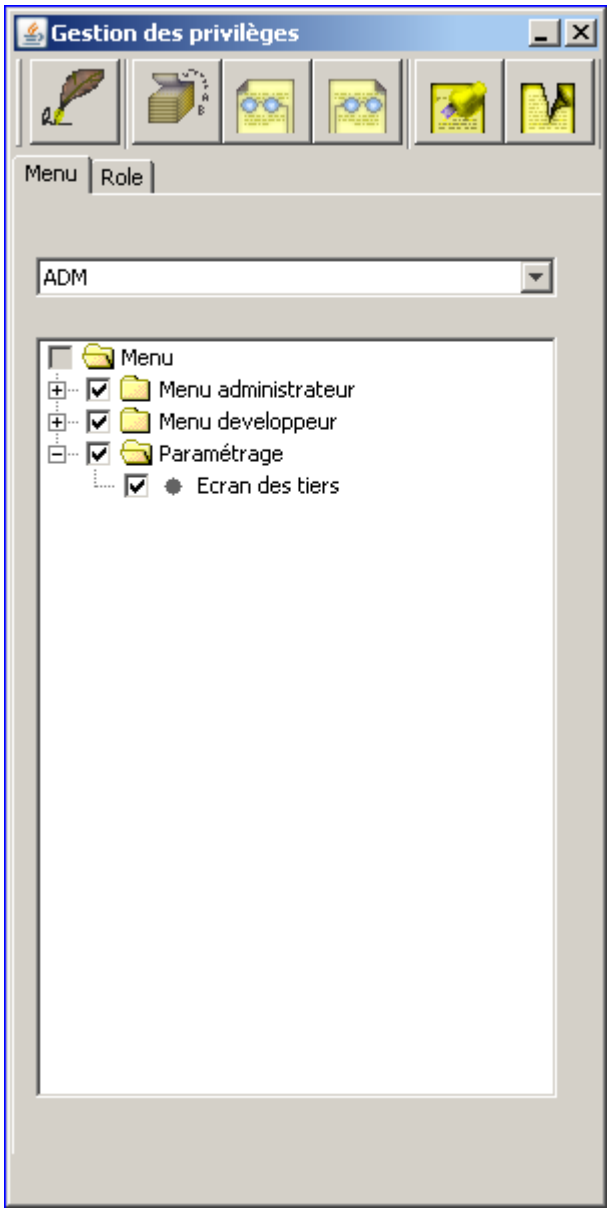
5.3.1.4 Modification des privilèges

Nous allons maintenant ouvrir l'écran de gestion des privilèges :

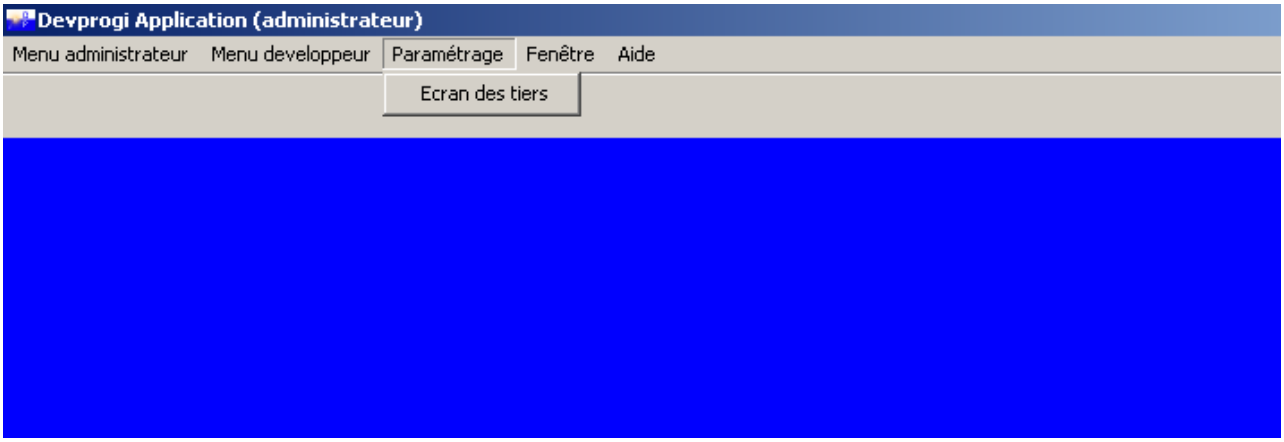
Menu administrateur / Gestion des privilèges



Cet écran permet d'afficher les menus en fonctions du rôle de l'utilisateur. Cochez les cases correspondantes aux menus que vous voulez voir figurer et saugardez :



Maintenant, vous pouvez vous redémarrer l'application et Ôh miracle, votre menu figure dans la barre de menu.

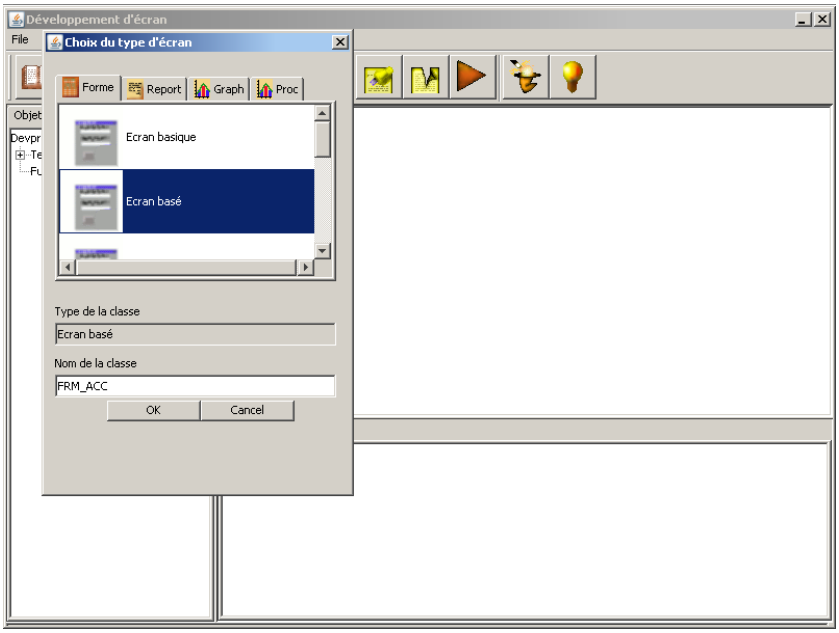


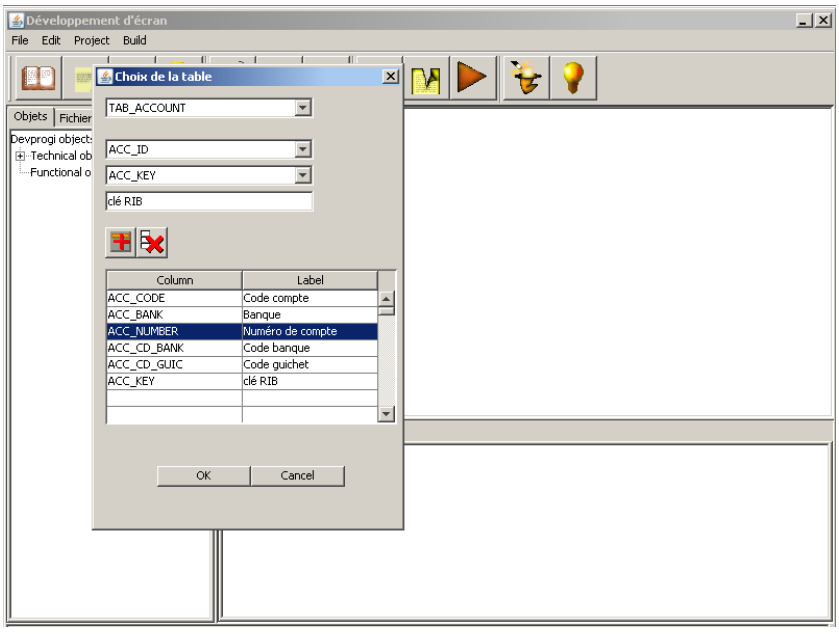
5.3.2 Ecran des catégories & écran des modes de paiement

Les écrans des catégories et des modes de paiement seront réalisés selon de le même principe que précédemment.

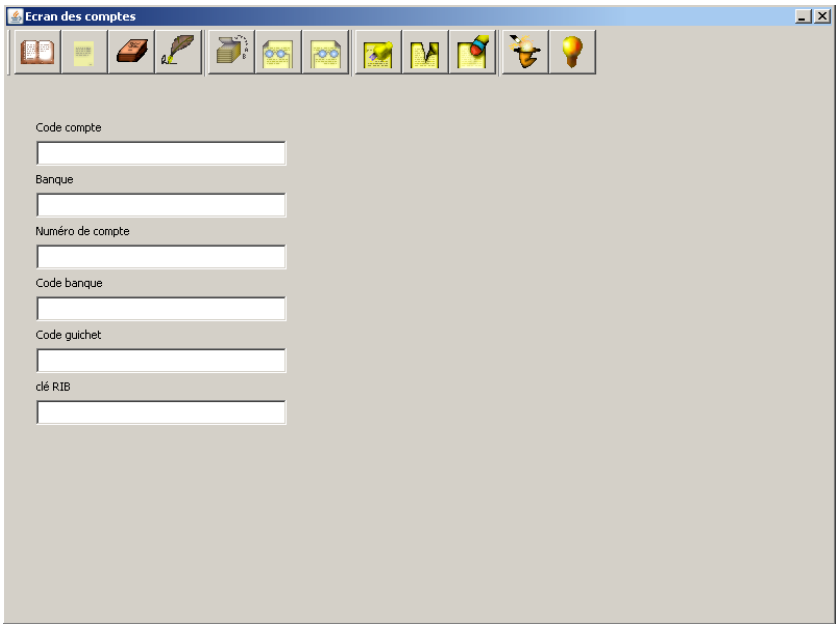
5.3.3 Ecran des comptes

Nous allons nous attaquer à l'écran des comptes. Cette fois, nous utiliserons un autre template.





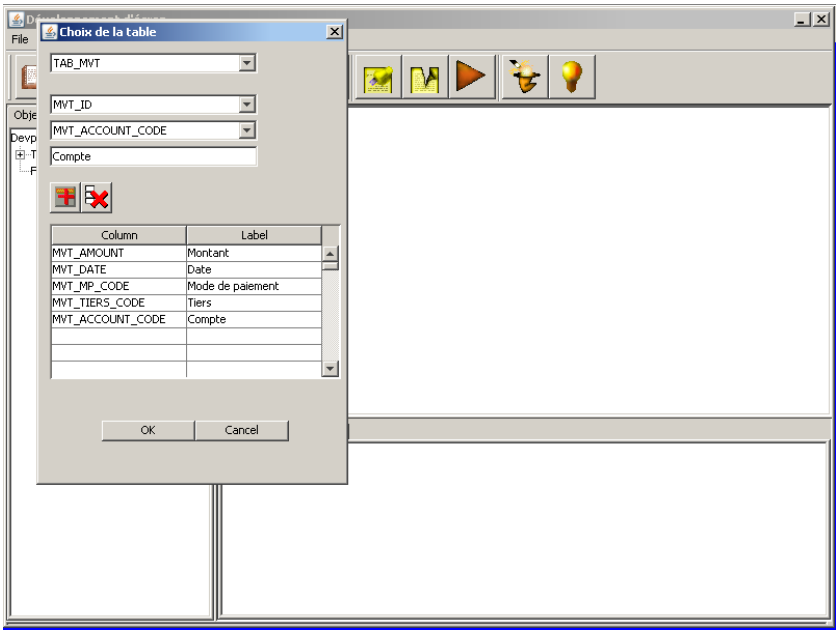
Après compilation, voici ce nous obtenons



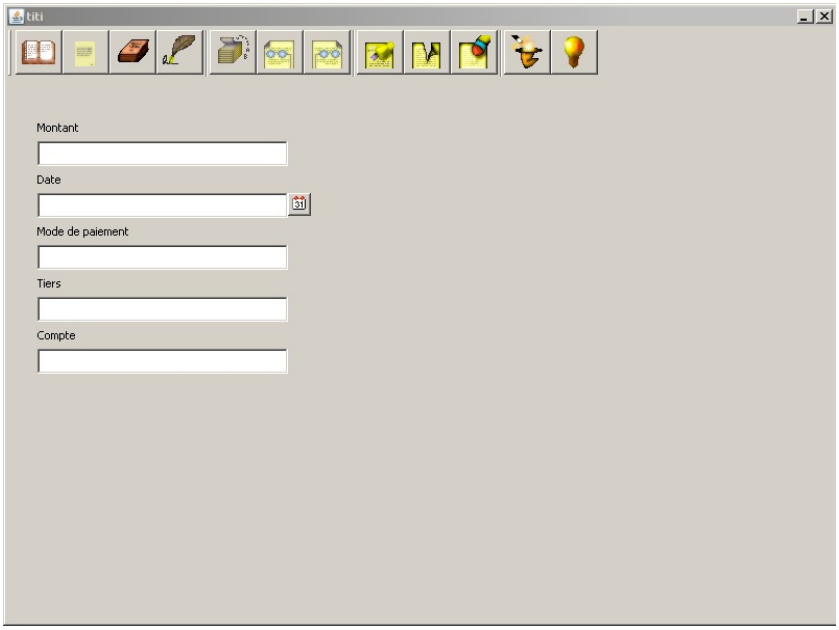
Reconnaissez que le design n'est pas folichon. A travers l'écran des mouvements, nous allons voir comment embellir un écran.

5.3.4 Ecran des mouvements

FRM_MVT

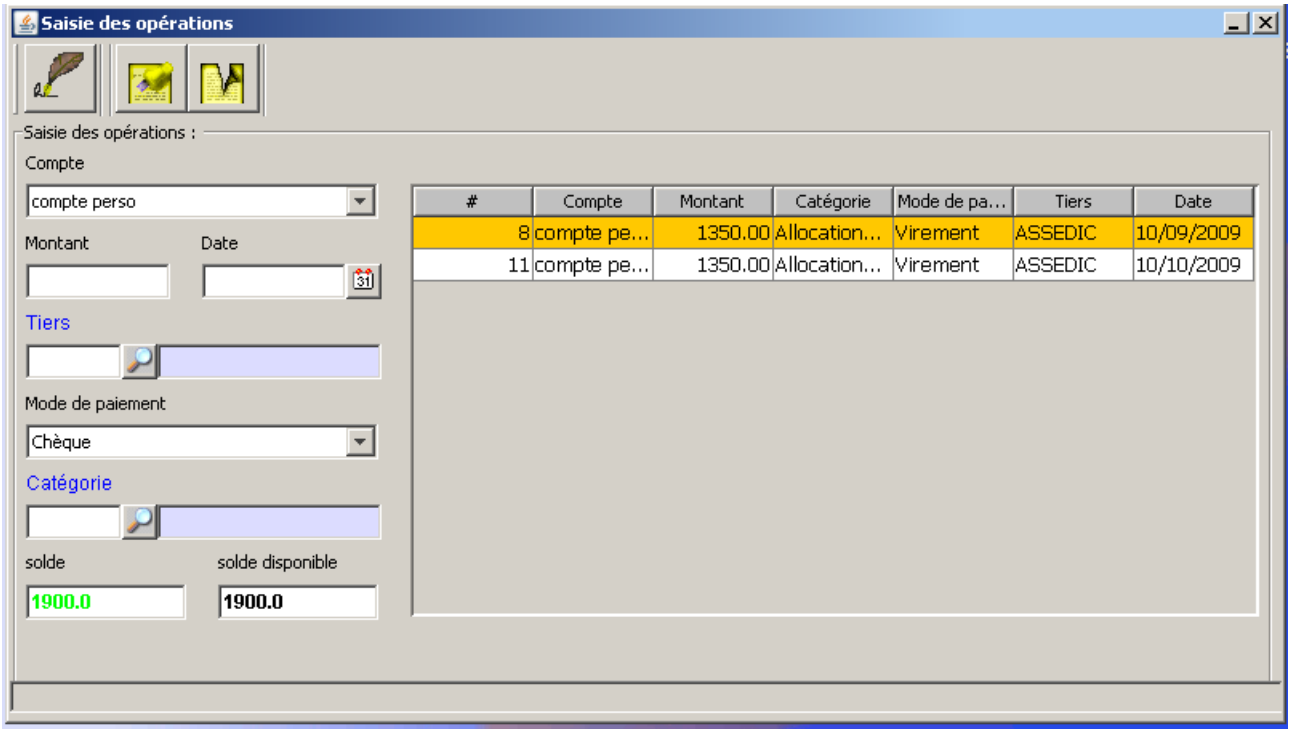


Tout comme pour les autres écrans, on clique sur OK, on compile, et on execute. Et voici le résultat !!!! pas joli, non?



Nous allons maintenant modifier le code de façon à donner une meilleur allure à l'écran. Certaines modifications se verront reporter dans le template.

A noter que dans la dernière application en cours de développement, l'écran de saisie des mouvements est



Bien que le modèle de données est plutot simple dans notre exemple, nous allons modifier le code de façon à se rapprocher de ce dernier.

Voici pour commencer le code que nous allons commencer à ajouter

```
 setFrmSize(800,450);
setPosition(100,0);
// apparence de la toolbar
openBt.setVisible(false);
newBt.setVisible(false);
closeBt.setVisible(false);
execQueryBt.setVisible(false);
nxtRecBt.setVisible(false);
prvRecBt.setVisible(false);
delRecBt.setVisible(true);
clearBt.setVisible(false);
wizardBt.setVisible(false);
```

```
HelpBt.setVisible(false);
```

La fonction setFrmSize(int dx,int dy) permet de redimensionner l'écran

La fonction setPosition(int dx,int dy) permet de repositionner l'ecran dans la fenetre MDI.

Nous avons maintenant la possibilité de redimensionner les champs.

```
setDimension("10,40,40,22", // Montant
             "10,100,30,22",  //Date
             "10,160,20,22",  // MP
             "10,220,20,22",  // TIERS
             "10,280,20,22"); // CPT
```

Enfin pour rendre plus agréable l'aspect visuel, nous pouvons rajouter une bordure.

```
setBorder( "Saisie des opérations : ");
```

Permet de rajouter une bordure

Maintenant, nous allons ajouter cette ligne de code.

Ajout d'un hyperlien

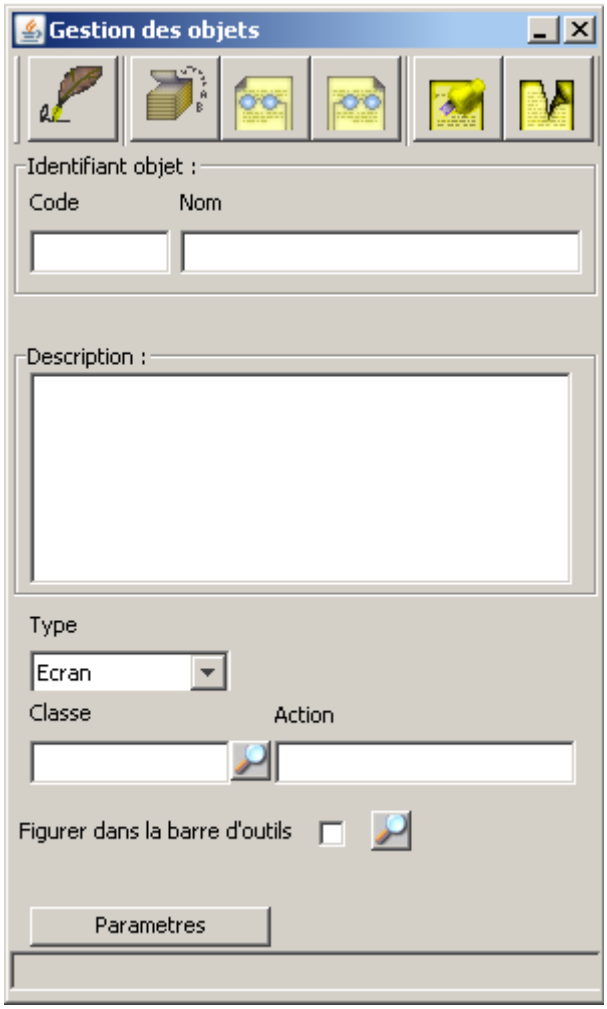
```
objLink lkti = new objLink("Tiers",
                           "OBJTIERS",
                           "OBJ",
                           getPanel(),10,195,100,22);
```

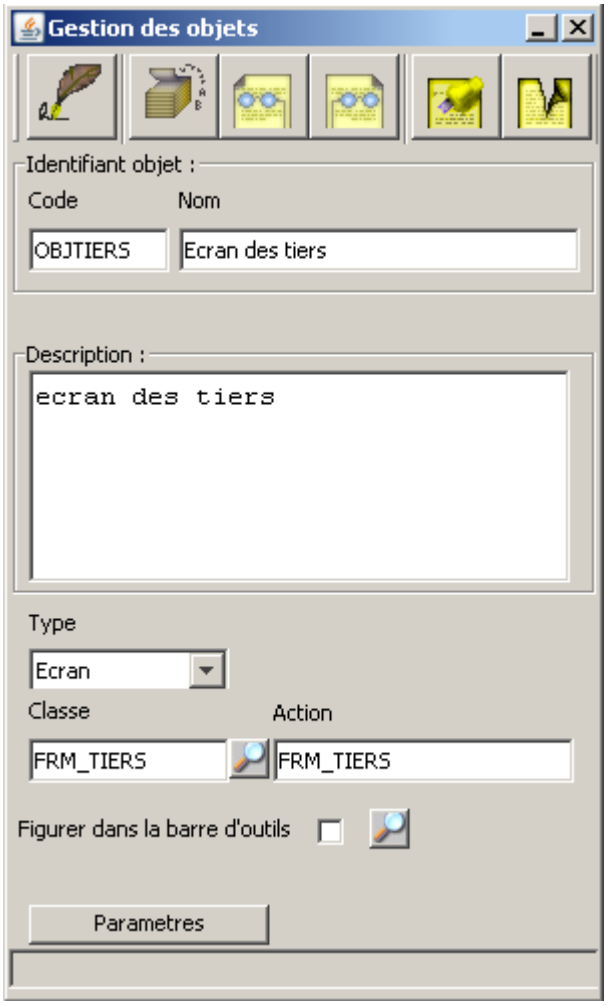
Avec comme ligne d'import

```
import devProgi.objGui.guiTool.objGui.*;
```

Cet hyperlien permettra d'appeler l'objet OBJTIER. Pour définir l'objet OBJTIERS, on se rend dans le menu

```
menu administrateur / Gestion des objets
```


Gestion des objets
Identifiant objet :
Code
Nom
Description :
Type
Ecran
Classe
Action
Figurer dans la barre d'outils
Parametres



Nous allons maintenant ajouter les listes de valeurs à l'écran.

```
	objButtonLov listTiers = new objButtonLov("Tiers", "TAB_TIERS", "TIERS_CODE" , "TIERS_NAME" );
	listTiers.setDescItemDisplayed(true);
	listTiers.setItem(getField("MVT_TIERS_CODE"));
	listTiers.addComponent(getPanel(),140);
	listTiers.addEventButtonLov(this);
```

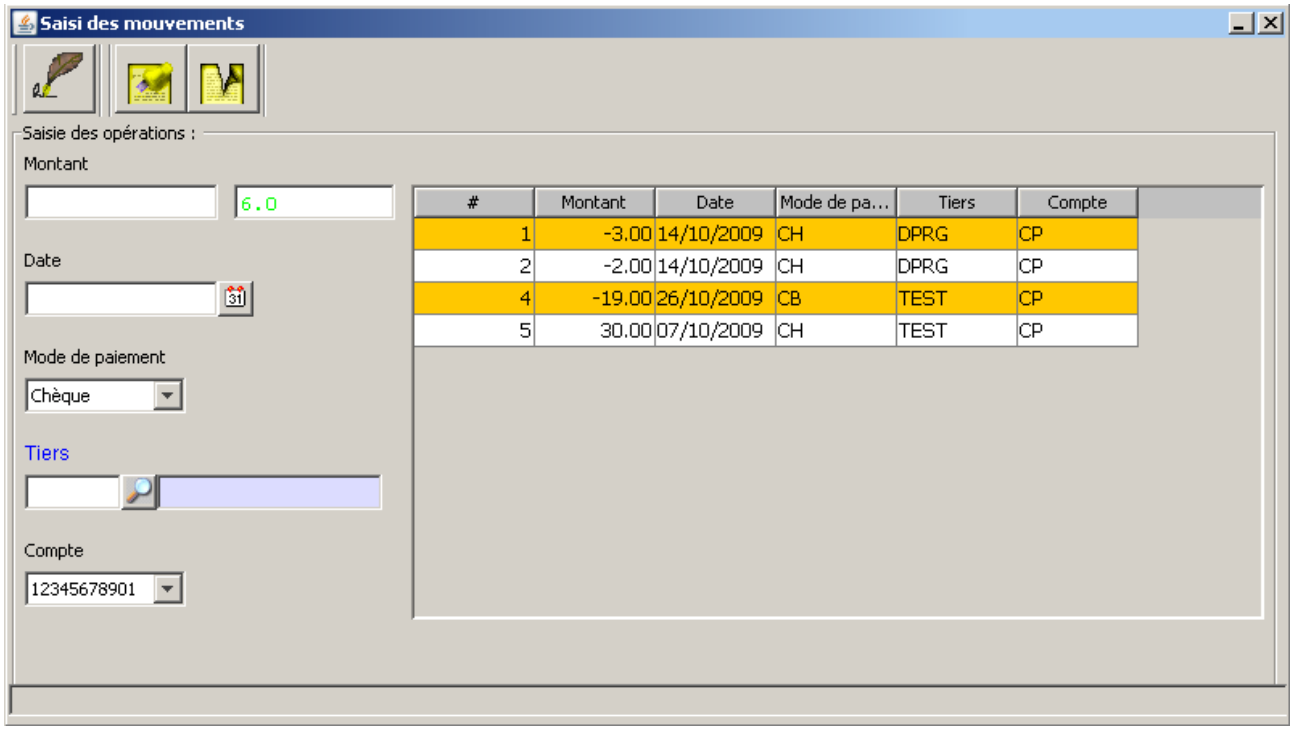
Le code complet vous maintenant présenté ci-dessous :

```
import devProgi.objGui.guiTool.objFrame.userForm;
import devProgi.objGui.guiTool.objGui.*;
import devProgi.objSystem.DbCnt.objDb;
import devProgi.objSystem.objEvent.*;
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.MouseEvent;
import javax.swing.JButton;

public class FRM_MVT extends userForm
implements EventButtonLov,              // Evenement sur la liste de valeur
           EventList,                   // Evenement sur la combobox
           EventTable                   // Evenement sur la table
{
    private objDbComboBox mp;
    private objDbComboBox ac;
    private objDbTable otsb;
    private objButtonLov listTiers;
    private objFieldArea txtSolde;

    public FRM_MVT()
    {
        // on donne un titre à l'écran
        super("Saisi des mouvements");

        // on dimensionne l'écran
        setFrmSize(800, 450);

        // on positionne l'écran
        setPosition(100, 0);

        // on rend disponible ou non les boutons de la toolbar
        openBt.setVisible(false);
        newBt.setVisible(false);
        closeBt.setVisible(false);
        execQueryBt.setVisible(false);
        nxtRecBt.setVisible(false);
        prvRecBt.setVisible(false);
        delRecBt.setVisible(true);
        clearBt.setVisible(false);
        wizardBt.setVisible(false);
        HelpBt.setVisible(false);

        // on positionne la table
        setTableName("TAB_MVT");

        // on donne l'identifiant de la table
        setColumnId("MVT_ID");

        // liste des champs de la table utilisé (chacun donne un champ de l'écran)
        setColumnName("MVT_AMOUNT",
                      "MVT_DATE",
                      "MVT_MP_CODE",
                      "MVT_TIERS_CODE",
                      "MVT_ACCOUNT_CODE");

        // libellé des champs utilisés
        setColumnLibel("Montant",
                       "Date",
                       "Mode de paiement",
                       " ",
                       "Compte");
```

```
        // dimension des champs
        setDimension("10,40,40,22",
                     "10,100,40,22",
                     "10,160,0,22",
                     "10,220,20,22",
                     "10,280,0,22");

        // construction du panel
        setPanel(buildForm());

        // ajout d'une bordure à l'écran
        setBorder("Saisie des opérations : ");

        // ajout d'un hyperlien
        objLink objlink = new objLink("Tiers",     // libellé de l'hyperlien
                                      "OBJTIERS", // objet sur lequel on pointe
                                      "OBJ",       // type d'hyperlien (ici, on pointe sur un objet)
                                      getPanel(), 10, 195, 100, 22);

        // création d'un bouton liste de valeur --> creation du bouton et du champs descriptif
        // objButtonLov(<titre de la fenetre>,<Table>,<colonne identifiant>,<colonne description>)
        listTiers = new objButtonLov("Tiers", "TAB_TIERS", "TIERS_CODE", "TIERS_NAME");
        listTiers.setDescItemDisplayed(true);       // faire apparaitre ou non le champ descriptif
        listTiers.setItem(getField("MVT_TIERS_CODE"));  // champ auquel on rattache la liste
                                                        // de valeur
        listTiers.addComponent(getPanel(), 140);
        listTiers.addEventButtonLov(this);              // associé les évenements

        // ajout d'une combobox (liste deroulante) pour le mode de paiement
        mp = new objDbComboBox(getPanel(), "select MP_NAME,MP_CODE from TAB_MP", 10, 160, 100, 22);
        mp.addListListener(this);

        // ajout d'une combobox (liste deroulante) pour le mode de paiement
        ac = new objDbComboBox(getPanel(),
                               "select ACC_NUMBER,ACC_CODE from TAB_ACCOUNT",
                               10, 280, 100, 22);
        ac.addListListener(this);

        // ajout d'un tableau
        otsb = new objDbTable(getPanel(), 250, 20, 530, 270);
        otsb.addEventTable(this);
        // positionnement du tableau sur une table
        otsb.setTableName("TAB_MVT");

        // positionnement de l'identifiant de la table
        otsb.setColumnId("MVT_ID");

        // colonnes utilisées
        otsb.setColumnName("MVT_ID",
                           "MVT_AMOUNT",
                           "MVT_DATE",
                           "MVT_MP_CODE",
                           "MVT_TIERS_CODE",
                           "MVT_ACCOUNT_CODE");

        otsb.reLoadTable();
        // libellé des colonnes du tableau
        otsb.setHeaderColumn("#",
                             "Montant",
                             "Date",
                             "Mode de paiement",
                             "Tiers",
                             "Compte");


        // initialisation des champs
        getField("MVT_MP_CODE").setText(mp.getValue());
```

```
        getField("MVT_ACCOUNT_CODE").setText(ac.getValue());

        // ajout d'un champ non basé pour le solde
        txtSolde = new objFieldArea(getPanel(), 140, 40, 100,22);
        // valorisation du champ txtSolde
        setSolde(ac.getValue());
    }

    public void whenLovClosed(ActionEvent actionevent)
    {
    }

    public void whenLovCalled(ActionEvent actionevent)
    {
    }

    public void whenListChanged(ActionEvent actionevent)
    {
        if(actionevent.getSource() == mp)
            getField("MVT_MP_CODE").setText(mp.getValue().trim());
    }

    public void whenTableRowSelected(MouseEvent mouseevent)
    {
        setWhereClause("MVT_ID = " + otsb.getValueAt(otsb.getSelectedRow(),0));
        execQuery();
    }

    public void whenTableRowModified(ActionEvent actionevent)
    {
    }

    public void userSave()
    {
        otsb.reLoadTable();
        otsb.setHeaderColumn("#",
                             "Montant",
                             "Date",
                             "Mode de paiement",
                             "Tiers",
                             "Compte");
        setSolde(ac.getValue());
    }

    public void userNewRec()
    {
        getField("MVT_MP_CODE").setText(mp.getValue());
        getField("MVT_ACCOUNT_CODE").setText(ac.getValue());
    }

    public void userExecQuery()
    {
        loadData();
    }

    public void userNextRec()
    {
        loadData();
    }

    public void userPrevRec()
    {
        loadData();
    }
```

```
  public void userDelRec()
 {
      setSolde(ac.getValue());
      otsb.reLoadTable();
        otsb.setHeaderColumn("#",
                             "Montant",
                             "Date",
                             "Mode de paiement",
                             "Tiers",
                             "Compte");


  }




    public void loadData()
    {
        listTiers.displayDescValue(getField("MVT_TIERS_CODE").getText());
        ac.setValue(getField("MVT_ACCOUNT_CODE").getText());
        mp.setValue(getField("MVT_MP_CODE").getText());
    }



    public void setSolde(String ac)
    {
        float f = objDb.getObjectFloat("select truncate(sum(IFNULL(MVT_AMOUNT,0)),2) from TAB_MVT
where MVT_ACCOUNT_CODE = '" + ac + "'");

        if(f <= 0)
        {
            txtSolde.setForeground(Color.red);
        }
        else
        {
            txtSolde.setForeground(Color.green);
        }

        txtSolde.setText(Float.toString(f));
    }

}
```


Saisi des mouvements
Saisie des opérations :
Montant
6.0
Date
Mode de paiement
Chèque
Tiers
Compte
12345678901
Montant Date Mode de pa... Tiers Compte
1 -3.00 14/10/2009 CH DPRG CP
2 -2.00 14/10/2009 CH DPRG CP
4 -19.00 26/10/2009 CB TEST CP
5 30.00 07/10/2009 CH TEST CP

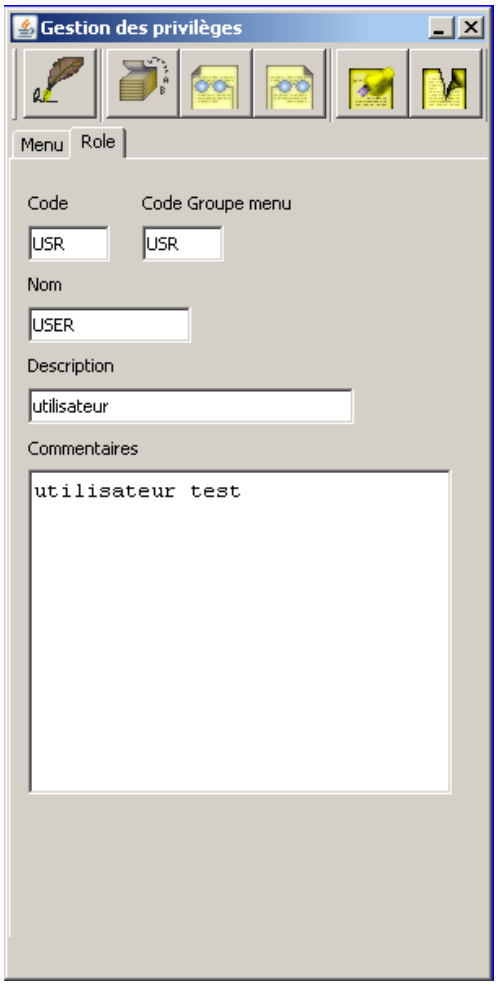
5.4 Finalisation de l'application

Maintenant, il faut pouvoir exporter cette application. Il est clair que lorsque l'utilisateur lancera l'application, il ne doit ni forcément se connecter, ni voir le menu développeur. Enfin, il ne doit pas figurer les classes et les sources de l'application.

C'est ce que nous allons voir. Cependant, il vous faut faire appel à quelques outils extérieur.

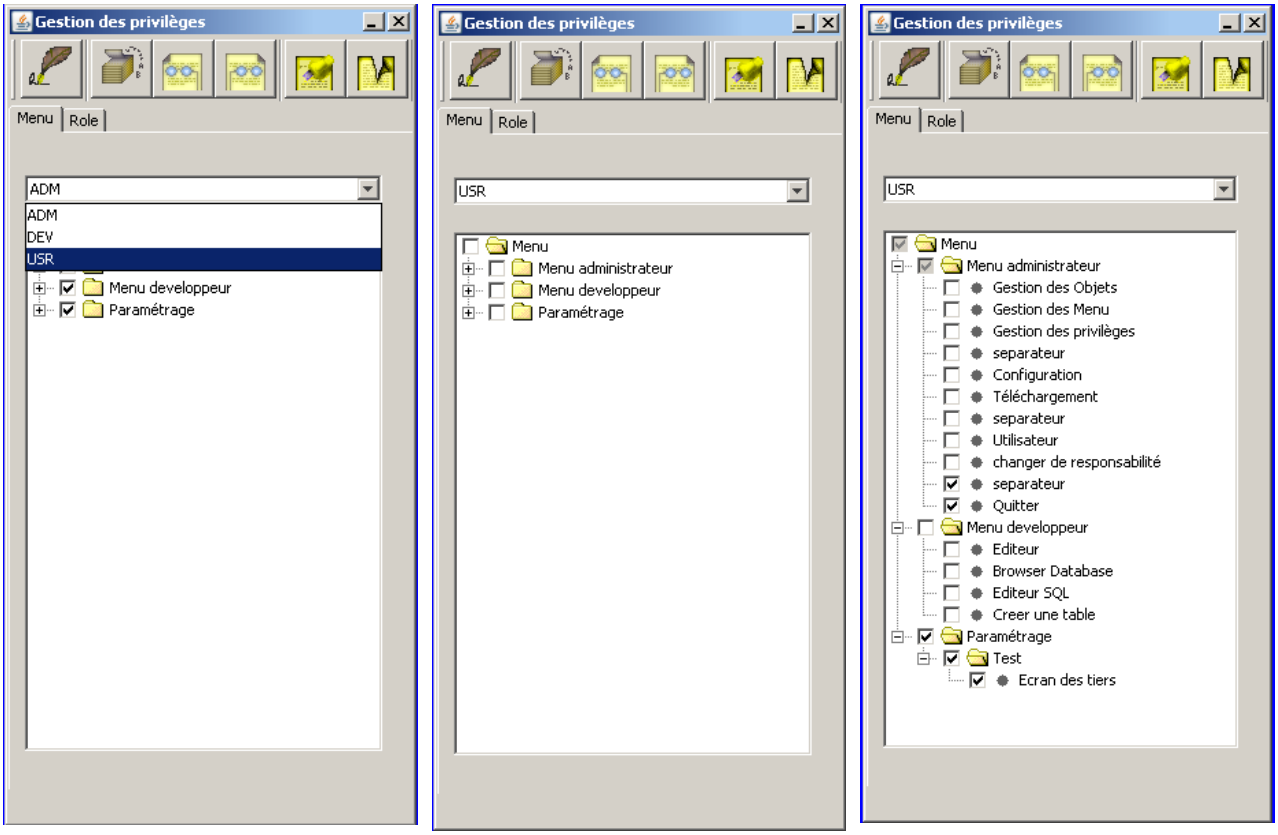
5.4.1 Création d'un rôle

Avant toute chose nous devons créer un rôle et lui attribuer les différents privilèges



5.4.2 Modification des privilèges

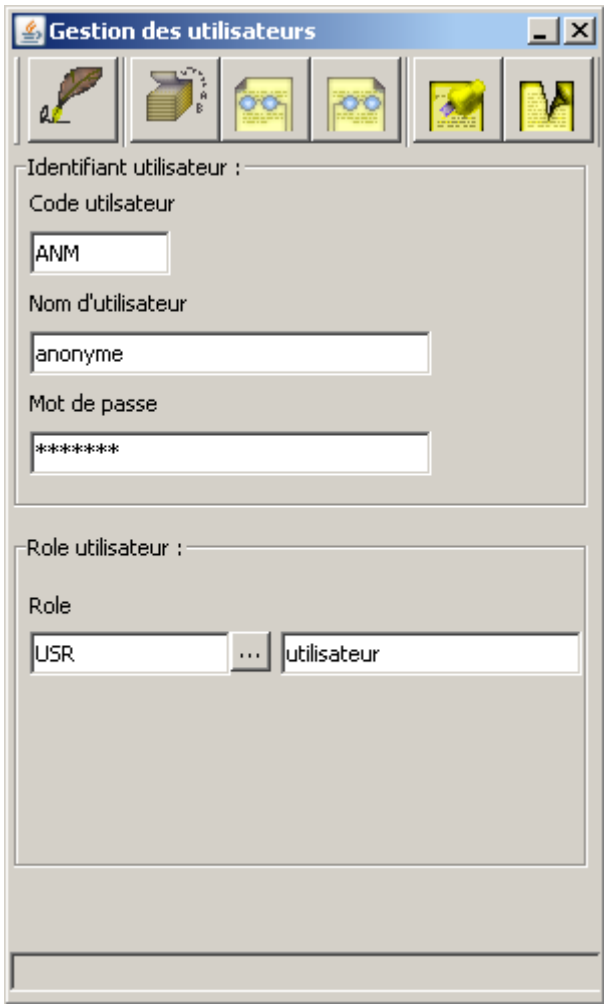
Une fois le nouveau rôle créé, vous choisissez les différents menu nécessaire à tous les utilisateurs ayant ce rôle.



5.4.3 Création d'un utilisateur

Vous pouvez désormais créer un utilisateur qui n'aura acces qu'au menu applicatif. Pour ce faire, vous le créer dans l'écran

Menu administrateur / Utilisateur



5.4.4 Création d'un jar

Maintenant, nous arrivons à la fin de l'application, enfin!!!

```
C:\devprogi\bin>"C:\Program Files\Java\jdk1.6.0_14\bin\jar" cvf applidev.jar *.class
```

et copiez le jar dans le répertoire dbdrivers (oui historiquement, si on peut dire, on je n'y mettais que les jar concernant les connexions aux différentes bases de données).

Dans un très proche futur (dans quelques jours même), j'implémenterai un écran pour préparer l'éxécutable.

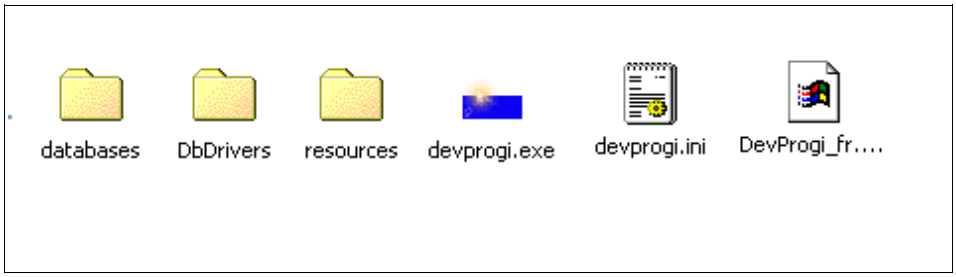
5.4.5 modification du fichier ini.

Pour le fichier ini, voici ce qui est nécessaire :

```
username=dpuser
password=dppwd
dbname=dpdb
driver=in.co.daffodil.db.jdbc.DaffodilDBDriver
url=jdbc:daffodilDB_embedded:dpdb;path=C:/devprogi/databases;create=false
mdi_toolbar=false
mdi_scrollbar=false
logon=no
default_user=anonyme
default_password=anonyme
class_path=DbDrivers/devprogi.jar
url_dl=http://www.devprogi.com/product/client/
```

5.4.6 finalisation de l'application

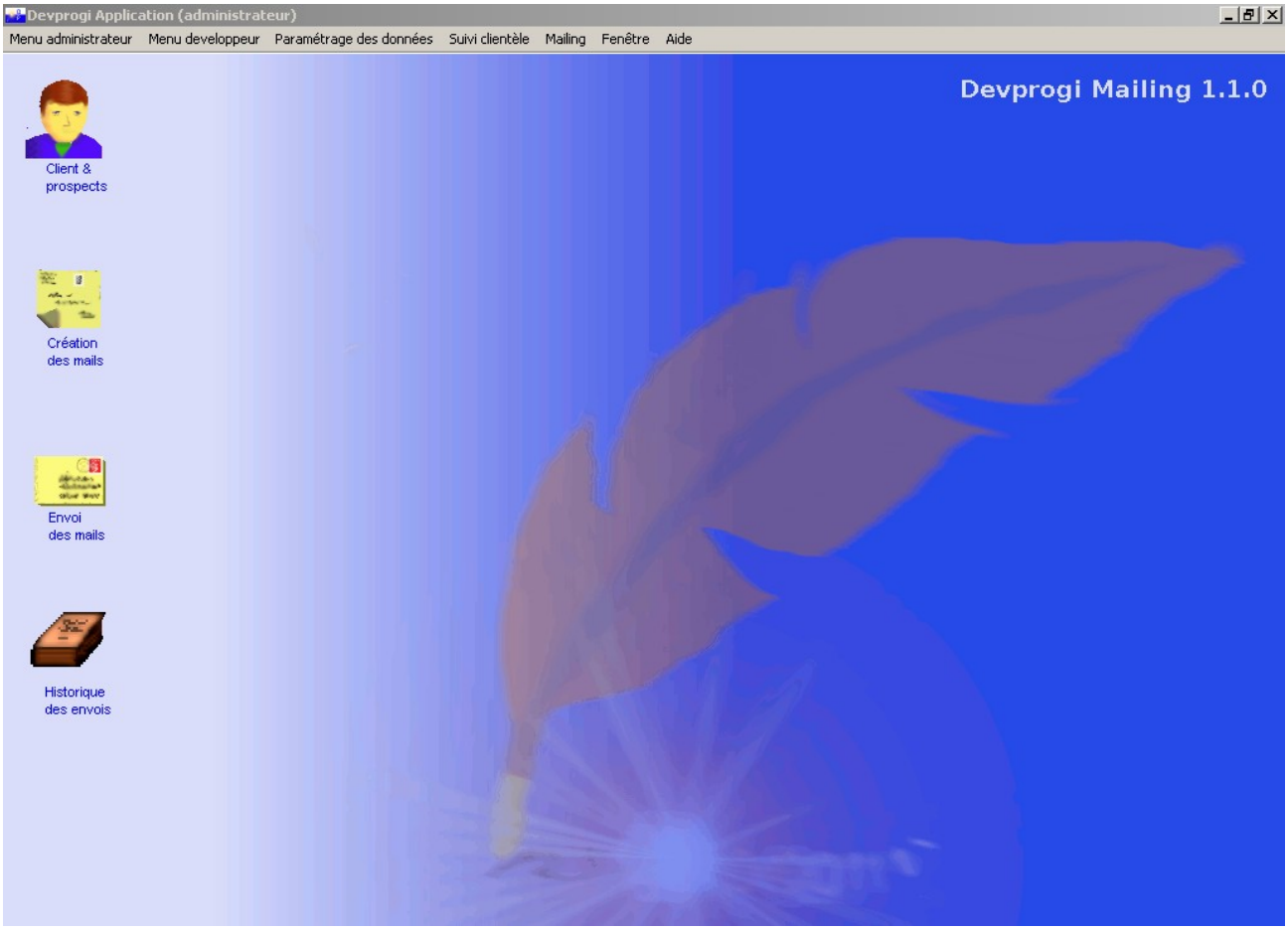
Voilà, nous sommes enfin arrivé à la fin. Dans cette dernière partie, je vous montre les fichiers et les répertoires à garder pour faire le fichier d'installation.



6 Autre exemple d'applications

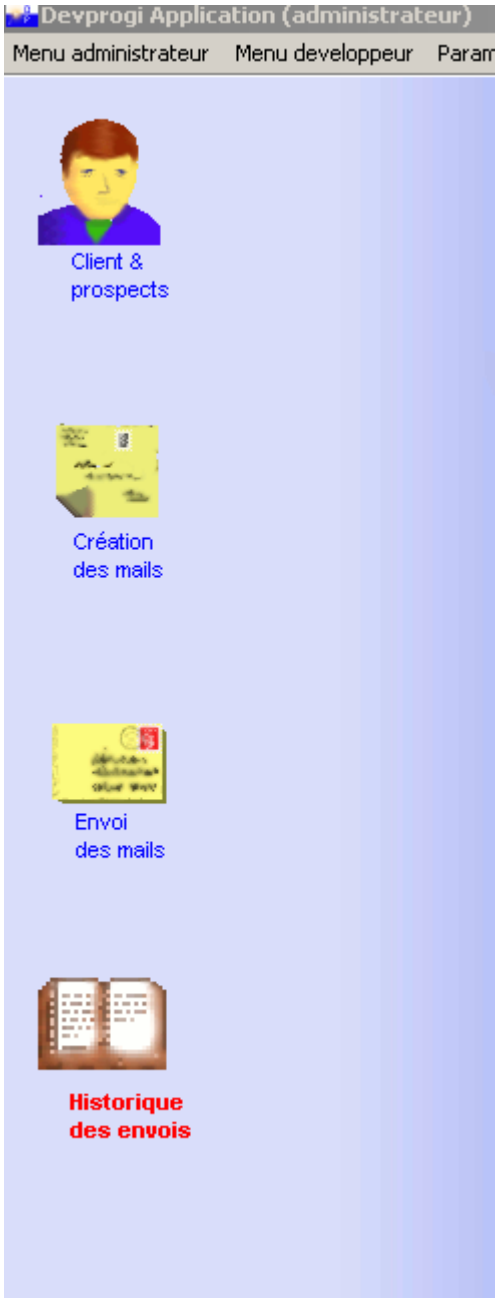
6.1 Devprogi Mailing 1.1.0

Devprogi Mailing est une application mettant en oeuvre la fonction d'envoi de mail. Par ailleurs, j'ai configuré l'application de sorte que je puisse avoir un fond différent et par ailleurs, utiliser des images en guise de boutons.



Pour info, les images sur le coté ont le comportement d'un bouton. Lorsque le pointeur de la sourie se trouve sur une image, elle est modifiée, et lorsqu'on clique dessus, elle se modifie aussi (voir images page précédente).


Devprogi Application (
Menu administrateur
Menu
Client &
prospects
Création
des mails
Envoi
des mails
Historique
des envois


Devprogi Application (administrateur)
Menu administrateur
Menu developpeur
Param
Client &
prospects
Création
des mails
Envoi
des mails
Historique
des envois