

GEST-D-201 – Informatique pour la gestion II

Semaine intensive 2007-2008

Prof. : Hugues Bersini

17-22/12/2007

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, ainsi que les fondements des applications web.

Projet

Durant cette semaine intensive, vous aurez à développer une petite application destinée à gérer les opérations en bourse de quelques clients fortunés. Votre programme interagira avec une base de données pour en extraire les informations et recalculer les cours des actions et obligations, et vous développerez un site web sommaire qui permettra de passer des ordres d'achat ou de vente.

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

A réaliser	Théorie liée	Objectif
Schéma relationnel de la BD	Les bases de données relationnelles (Rappel)	Lundi Matin
Diagramme de classes	Programmation OO en Java et UML	Mardi midi
Application v1 : OO	Programmation OO en Java	Mercredi Soir
Application v2 : JDBC	Connexions aux BD en Java avec JDBC	Jeudi Soir
Site Internet : PHP	HTML et PHP	Samedi Midi

Enoncé

Vous devez réaliser un programme permettant de gérer les ordres d'achats et de ventes d'actifs sur un marché boursier. Les ordres sont passés par des clients (caractérisés par leur nom et adresse). Chaque client possède un compte en banque duquel sont prélevés ou sur lequel sont versés les montants des ordres effectués. La bourse pour laquelle vous travaillez gère deux types d'actifs : des actions et des obligations. Tout actif est associé à une certaine valeur principale P (fixe) et se traite à un certain cours C (variable), qui se détermine différemment suivant le mode de valorisation propre à chaque type d'actifs : le cours d'une action se calcule en fonction du dividende D et du nombre de titres achetés (A) et vendus (V), suivant la formule :

$$C_a = \frac{(P + D) \times (1 + \ln(A/V))}{1 + r}$$

Où r représente le taux d'intérêt (fixé par le marché). Les obligations, elles, se valorisent en fonction de la valeur de leur coupon C et du taux d'intérêt du marché, suivant la formule :

$$C_o = \frac{P + C}{1 + r}$$

Votre programme Java devra permettre de consulter les clients, les ordres effectués et les différents actifs et de réactualiser leur cours, tandis que votre site Web permettra aux clients de passer leurs ordres (achats et ventes) par Internet.

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

Vous ne devrez pas créer physiquement la base de données, mais vous utiliserez une base créée par nos soins sur un serveur. Le moment venu, votre programme Java de même que votre site Internet devront se connecter à cette base de données commune à tous les groupes.

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 « Bourse » contenant une méthode « main »
4. Dans cette méthode « main », instanciez et affichez à l'écran quelques objets de chaque classe, créez quelques ordres d'achats et de ventes, et affichez sur la console le détail de chaque actif après avoir recalculé tous les cours.
5. Lancez votre programme pour le tester (cf. ci-dessous)
6. Enrichissez ensuite votre méthode « main » pour qu'elle stocke les ordres et les actifs dans des collections « ArrayList » en rendant OO la classe Bourse.
7. Ordonnez l'affichage sur la console de tous les objets des collections « ArrayList » en veillant à exploiter au maximum le polymorphisme.
8. Lancez à nouveau votre programme pour vous assurer qu'il fonctionne.

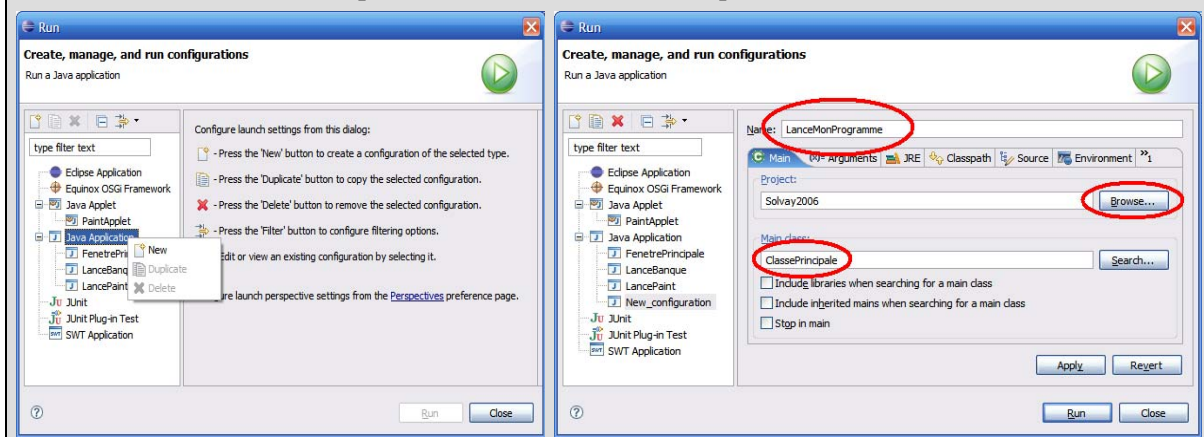
Encadré 1 : Exécuter une application Java

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) :

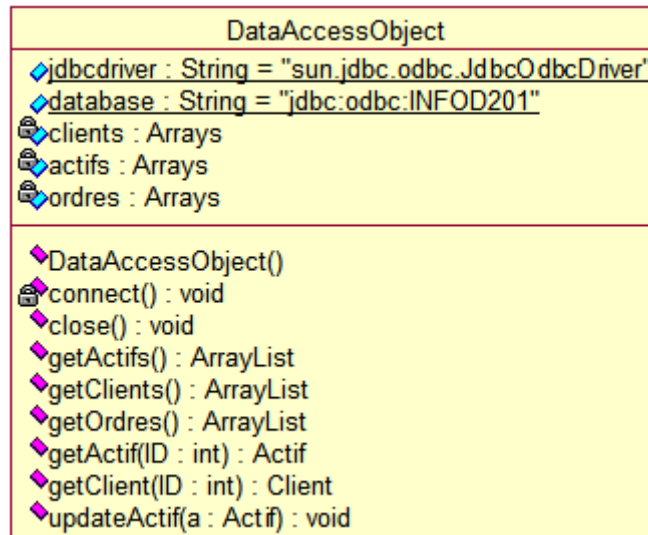
1. Dans le menu « Run », cliquez sur l'option « Run... »
2. Cliquez avec le bouton droit sur « Java Application » puis « New »
3. Donnez un nom à la configuration, indiquez celui de votre projet (si ce n'est fait) et de votre classe principale, puis cliquez sur le bouton Run.
4. Désormais, pour démarrer,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



5. Application V2 - JDBC

L'objectif de cette étape est de connecter votre programme à la base de données pour qu'il en extraie les informations et y mette à jour les cours. A la fin de cette étape, votre programme devra au minimum pouvoir afficher les détails des actifs et remettre à jour leur cours dans la DB et idéalement aussi pouvoir afficher les clients et les ordres effectués :

1. Créez une nouvelle classe « DataAccessObject » qui sera chargée de communiquer avec la DB. Le diagramme UML de cette classe vous est proposé ci-dessous:



Encadré 2 : Squelette de la classe DataAccessObject

La structure et le code de base de cette classe vous sont fournis ci-dessous. Veillez 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:INFOD201";
    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 3 : Envoyer une requête update à la base de données

Pour envoyer une requête update 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 = "UPDATE assets SET asset_currentprice="+XXX+" WHERE
asset_id ="+YYY+";";
    stmt.executeUpdate(query);
    stmt.close();
} catch(Exception e){
    e.printStackTrace();
}
```

Encadré 4 : 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 (exemple pour les actifs) :

```
try {
    if(conn==null)
        this.connect();
    Statement stmt = conn.createStatement();
    String query = "SELECT * FROM assets;";
    ResultSet rs = stmt.executeQuery(query);
    while(rs.next()){
        int a_id = rs.getInt("asset_id");
        String a_n = rs.getString("asset_name");
        String a_t = rs.getString("asset_type");
        double a_p = rs.getDouble("asset_principal");
        double a_c = rs.getDouble("asset_currentprice");
        double a_div = rs.getDouble("asset_dividend");
        double a_cp = rs.getDouble("asset_coupon");
        if(a_t.equalsIgnoreCase("stock"))
            new Stock(a_id,a_n,a_p,a_c,a_div);
        else
            new Bond(a_id,a_n,a_p,a_c,a_cp);
    }
    rs.close();
    stmt.close();
} catch(Exception e){
    e.printStackTrace();
}
```

Encadré 5 : Requêtes utiles pour le programme Java

Voici les requêtes dont vous pourriez avoir besoin :

- SELECT * FROM assets ORDER BY asset_name;
- SELECT * FROM clients ORDER BY client_nom;
- SELECT * FROM orders ORDER BY order_client, order_asset, order_id;
- UPDATE assets SET asset_currentprice='XXX' WHERE asset_id = 'YYY' ;

2. Une fois cette nouvelle classe réalisée, vous remplacerez le code de votre ancienne méthode « main » (qui créait des actifs arbitrairement) par un code permettant de récupérer les actifs de la DB (idéalement aussi les clients et les ordres) en exploitant la classe « DataAccessObject » ci-dessus. Votre main recalculera ensuite tous les cours et les mettra à jour dans la base de 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
    // (actifs, clients, ordres)

    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 extraire les actifs

    // A FAIRE PAR VOUS: Définir des méthodes pour mettre à jour les
    // cours dans la base de données.
}
```

6. Site Internet (PHP)

La dernière étape du projet qui vous est demandée consiste à créer un petit site Internet sommaire permettant aux clients d'effectuer des opérations (achat ou vente) d'actifs sur le marché. L'objectif minimum consiste à réaliser un formulaire web permettant de sélectionner un client dans une liste déroulante, de sélectionner un actif dans une liste déroulante, d'indiquer une quantité à vendre ou à acheter et de préciser s'il s'agit d'un achat ou d'une vente. Votre formulaire pourrait donc ressembler à ceci :

Idéalement, votre site devrait attendre que l'utilisateur introduise son login et son mot de passe afin de lui permettre d'acheter ou de vendre sous condition. Dans cette optique, l'utilisateur ne devra plus être choisi dans une liste, et l'opération – avant d'être enregistrée dans la base de données, devra être validée : en cas d'achat il faudra veiller à ce que le compte du client soit suffisamment approvisionné et en cas de vente il faudra vérifier que l'utilisateur possède bien les actions ou obligations qu'il essaie de vendre.

Encadré 6 : Requêtes utiles pour le site PHP

Voici les requêtes dont vous pourriez avoir besoin pour votre site PHP :

```

- SELECT * FROM clients ORDER BY client_nom;
- SELECT * FROM assets ORDER BY asset_name;
- SELECT asset_currentprice FROM assets WHERE asset_id='XXX';
- SELECT client_accountbalance FROM clients WHERE client_id='YYY';
- SELECT SUM(order_amountpurchased) np, SUM(order_amountsold) ns FROM
orders WHERE orders.order_client='YY' AND orders.order_asset='XXX';
- INSERT INTO orders VALUES('','YYY','XXX','ZZZ','0','PPP');
- INSERT INTO orders VALUES('','YYY','XXX','0','ZZZ','PPP');
- UPDATE clients SET client_accountbalance='WWW' WHERE client_id='XXX';
- SELECT * FROM clients WHERE client_login='LLL' AND client_pw='NNN';"

```

Encadré 7 : Code PHP pour établir une connexion à la base de données

Voici le code nécessaire pour connecter vos pages PHP à la base de données :

```
$base_database = "BASEDEDONNEES";
$login_database = "LOGIN_DB";
$password_database = "PW_DB";
$db = mysql_connect("dev.ulb.ac.be", $login_database, $password_database);
if(!mysql_select_db($base_database,$db)) {
    print "Erreur ".mysql_error()."<br>";
    mysql_close($db);
    exit;
}
```

Encadré 8 : Code PHP pour envoyer une requête à la base de données

Voici le code PHP nécessaire pour envoyer une requête action à la base de données :

```
mysql_query("CODE DE LA REQUETE SQL", $db);
```

Voici le code PHP nécessaire pour envoyer une requête sélection à la base de données et parcourir les lignes renvoyées (*Result Set*) :

```
$resultset = mysql_query("CODE DE LA REQUETE SQL", $db);
for($i=0; $i<mysql_numrows($resultset); $i++) {
    $a_id = mysql_result($resultset,$i,"asset_id");
    $a_n = mysql_result($resultset,$i,"asset_name");
    $a_t = mysql_result($resultset,$i,"asset_type");
    $a_c = mysql_result($resultset,$i,"asset_currentprice");
    print "a_id - $a_n - $a_t - $a_c<br>";
}
```