Java DataBase Connections

JDBC est une api
permettant l'utilisation de bases de données
indépendamment
du type de SGBD utilisé.

les requêtes sont fournies en SQL standard.

JDBC

Utilisation de plusieurs classes et interfaces du package java.sql

java.sql.DriverManager
Une classe d'utilitaire permettant en particulier
d'obtenir une connexion à la base de données

Des interfaces, en particulier :

java.sql.Connection

java.sql.Statement : les requêtes

java.sql.ResulSet : les résultats des requêtes

Les différentes étapes pour utiliser une base de données

•1 Charger la classe correspondant au driver souhaité

(réalisé parfois "automatiquement")

```
Class.forName("oracle.jdbc.driver.OracleDriver")
```

•2 Obtenir une connexion à la base de données

```
Connection connexion = DriverManager.getConnection(…,…,…);
```

•3 Construire et exécuter une requête

```
Statement stmt=connexion.createStatement( );
ResultSet rs = stmt.executeQuery("…");
ou
int count = stmt.executeUpdate("…");
```

•4 Traiter le résultat obtenu

en utilisant les méthodes de ResultSet

.5 Fermer le statement et la connexion

```
connexion.close( );
```

1. Charger le bon driver jdbc

Le driver JDBC est une classe implémentant l'interface java.sql.Driver.
Elle est spécifique à un SGBD précis.

Pour indiquer le driver utilisé, on utilise la méthode **Class.forName**.
Elle permet le chargement dynamique de classe
(nous revenons en détail sur ce point dans la suite du cours).

exemple de chargement de driver :

```
Class.forName("com.mysql.jdbc.Driver");// pour mysql
Class.forName("oracle.jdbc.driver.OracleDriver");// pour oracle
Class.forName("jdbc.odbc.JdbcOdbcdriver");// pour odbc
Class.forName("org.postgresql.Driver"); // pour postgresql
Class.forName("org.sqlite.JDBC"); // pour sqlite
```

2. Se connecter à la base de données

Connexion connexionAvecMysql =
DriverManager.getConnection("jdbc:mysql://localhost/pary","root","aw3peZ");

La syntaxe des urls

url avec Postgres : "jdbc:postgresql://urlServeur/nomDeMaBase"

url avec mysql : "jdbc:mysql://urlServeur/nomDeMaBase”

url avec sqlite : "jdbc:sqlite:fileName”

```
class DriverManager{
.../** Attempts to establish a connection to the given database URL. The
* DriverManager attempts to select an appropriate driver
* from the set of registered JDBC drivers.
* @ param url - a database url of the form jdbc:subprotocol:subname
* @ param user - the database user on whose behalf the connection is being made
* @ param password - the user's password
* @ return  a connection to the URL
* @ throw  SQLException if a database access error occurs */
public static Connection getConnection(String url,String user,String  password)
                throws SQLException {…}
```

Problèmes classiques lors des premières utilisation de jdbc

Les problèmes classiques avant d'arriver à exécuter un programme utilisant jdbc :

- la classe du driver et les classes associées (regroupées dans un jar) doivent être dans le classpath au moment de l'exécution

- l'url de la base doit être écrite correctement

- l'user et son password doivent être enregistrés dans le SGBD …

- la base doit être utilisable à partir du poste où l'on exécute le programme Java

Exemple : se connecter à une base de données

```
public static Connection getConnection( ) throws SQLException{
String nomBase="pary";
String hostname="www.ens.math-info.univ-paris5.fr";
String urlBd = "jdbc:mysql://"+hostname+"/"+nomBase;
String user="pary",  password="xxxxxxxx";
return DriverManager.getConnection(urlBd,user,password );
}
public static void main(String [ ] args) {
try { Class.forName("com.mysql.jdbc.Driver");
 Connection connection = getConnection();
 System.out.println("Connexion réalisée !");
 connection.close( ); }
catch (SQLException e){
      System.out.println("Connexion non effectuée !");
      e.printStackTrace( ); }
catch (ClassNotFoundException e){
System.out.println("Vérifier que la classe du  driver est dans le classpath");
e.printStackTrace( );} }
```

3 exécution de requêtes

// tout d'abord, récupération d'un Statement à partir de la connexion
Statement stmt = connection.createStatement( );

// puis exécution de la requête en utilisant :
a) executeUpdate pour les requêtes de mise à jour de la base
INSERT, CREATE, DELETE ...

b) executeQuery pour les requêtes interrogeant la base
SELECT

executeUpdate

```
// tout d'abord, récupération d'un Statement à partir de la connexion
Statement stmt = connection.createStatement( );

// puis exécution de la requête
int count = stmt.executeUpdate(
"INSERT INTO QUESTION(intitule,reponse,auteur,niveau)"+
        "VALUES ('Les classes sont toujours abstraites','non', 'pary',1)"
 );
/** Executes the given SQL statement, which may be an INSERT, UPDATE, or DELETE statement
   *  or an SQL statement that returns nothing, such as an SQL DDL statement.
   * @param sql - an SQL INSERT, UPDATE or DELETE statement
   *                                or an SQL statement that returns nothing
   * @ return either the row count for INSERT, UPDATE or DELETE statements,
   *                or 0 for SQL statements that return nothing
   * @ throws SQLException - if a database access error occurs
                                       or the given SQL statement produces a ResultSet object
*/
public int executeUpdate(String sql)  throws SQLException
```

exemple de création de table avec executeUpdate

```
/** crée la table question en utilisant la 'connection' fournie*/
private static void createTableQuestion(Connection connection)
                                                    throws SQLException{
 Statement statement = null;
 try{statement = connection.createStatement( );
     String request = "CREATE TABLE question" +
                      " (intitule VARCHAR(80),reponse VARCHAR(80),"+
                      "auteur VARCHAR(80),niveau INTEGER not NULL)";

      statement .executeUpdate(request);
     }
 finally {if (statement !=null) statement.close( );}

}
```

exemple de création de table avec executeUpdate Try_with-ressources-statement

```
/** crée la table question en utilisant la 'connection' fournie*/
private static void createIfNotExistsTableQuestion(Connection connection) throws SQLException{

  String request = "CREATE TABLE question" +
                          " (intitule VARCHAR(80),reponse VARCHAR(80),"+
                           "auteur VARCHAR(80),niveau INTEGER not NULL)";
  try (Statement statement = connection.createStatement( ))

     {statement .executeUpdate(request);}
```

executeQuery

```
Statement stmt = connection.createStatement( );

ResultSet rs= stmt.executeQuery("SELECT intitule,reponse FROM question");
```

```
// Interface Statement
/** Executes the given SQL statement, which returns a single ResultSet object.
   * @param sql - an SQL statement to be sent to the database sql ,
      typically a static SQL SELECT statement
   * @ return    a ResultSet object that contains the data produced by the given query;
      never null
   * @ throws SQLException - if a database access error occurs
      or the given SQL statement produces anything other than a single ResultSet object
   */
public ResultSet executeQuery(String sql) throws SQLException{…}
```

4 Utiliser le ResultSet

```
ResultSet rs= stmt.executeQuery("SELECT intitule,reponse FROM question");
```

La méthode next de ResultSet permet de parcourir une à une les lignes du résultat.
La méthode getObject de ResultSet permet de récupérer la valeur d'une colonne sur la ligne courante.

```
while (rs.next( )){
Object obj1 = rs.getObject(1);// l'intitulé
Object obj2 = rs.getObject(2);// la réponse
System.out.println("("+obj1+","+obj2+")") ;
}
```

La méthode next permet de passer à la ligne suivante.
Elle rend false s'il n'y a pas de ligne suivante
Le premier appel à next permet de se positionner sur la première ligne

Quelques méthodes de l'interface java.sql.ResultSet

```
/** deplace le curseur vers la ligne suivante
    @return vrai si il y a une ligne suivante
    @exception SQLException si une erreur survient lors de l'accès à la base*/
public boolean next( ) throws SQLException

/** rend l'objet de la colonne n° 'numCol' de la rangée courante de ce ResultSet
@exception SQLException si une erreur survient lors de l'accès à la base*/
public Object getObject(int numCol) throws SQLException

/** rend l'objet de la colonne de nom 'nomCol' de la rangée courante de ce ResultSet
@exception SQLException si une erreur survient lors de l'accès à la base*/
public Object getObject(String nomCol) throws SQLException
```

Il existe aussi de nombreuses autres méthodes similaires getString, getDouble( ) …
utilisable pour certains types de données SQL

Quelques correspondances entre type SQL et type Java

SQL	classe de getObject( )	Autre fonction utilisable
CHAR	String	String getString(…)
VARCHAR	String	String getString(…)
INTEGER	Integer	int getInt(…)
REAL	Float	float getFloat(…)
DOUBLE	Double	double getDouble(…)
TIMESTAMP	java.sql.TimeStamp	TimeStamp getTimeStamp(…)

```
ResultSet rs= stmt.executeQuery("SELECT niveau,auteur FROM question");
while (rs.next( )){
int niveau = rs.getInt(1);
String auteur = rs.getString(2);
System.out.println("( niveau : "+niveau+" auteur :"+auteur+")") ;
}
```

extraits de la documentation en ligne de ResultSet

public interface **ResultSet**

A table of data representing a database result set, which is usually generated by executing a statement that queries the database.

A ResultSet object maintains a cursor pointing to its current row of data.

Initially the cursor is positioned before the first row.

The next method moves the cursor to the next row, and because it returns false when there are no more rows in the ResultSet object, it can be used in a while loop to iterate through the result set.

A default ResultSet object is not updatable and has **a cursor that moves forward only**. Thus, it is possible to iterate through it only once and only from the first row to the last row.

The ResultSet interface provides getXXX methods for retrieving column values from the current row. Values can be retrieved using either the index number of the column or the name of the column. In general, using the column index will be more efficient.

Columns are numbered from 1.

PreparedStatement

Avec createStatement

```
String srequete="SELECT intitule, reponse FROM question "+
        "WHERE question.niveau= 4 AND question.auteur='dupond' "

Statement stmt = connexion.createStatement( );
ResultSet result = stmt.executeQuery(srequete);
```

Avec preparedStatement

```
String srequete="SELECT intitule, reponse FROM question "+
        "WHERE question.niveau= ? AND question.auteur=? "

PreparedStatement stmt = connexion.prepareStatement(srequete);
stmt.setInt(1,4);//niveau = 4
stmt.setString(2,"dupond");// auteur = dupond
ResultSet result = stmt.executeQuery( );
```

L'interface ResultSetMetaData

ResultSetMetaData permet de connaître le nombre de colonnes, le nom des colonnes, ...

La méthode getMetaData de l'interface ResultSet permet de récupérer une instance de ResultSetMetaData.

```
/** @return le nombre de colonne du ResultSet de ce ResultSetMetaData*/
public int getColumnCount( ) throws SQLException  {…}

/**@return le nom d'affichage de la colonne n° 'column' de ce ResultSetMetaData
  ATTENTION : les colonnes sont numérotées à partir de 1 !*/
public String getColumnLabel(int column) throws SQLException

/** Get the designated column's specified column size.
    * For numeric data, this is the maximum precision.
    * For character data, this is the length in characters...
    */
public int getPrecision(int column) throws SQLException;
```

Transactions : commit et rollback

```
// par défaut, un commit est exécuté après chaque requête
// on demande ici que ce ne soit pas le cas
connection.setAutocommit(false);

// dans ce cas, la transaction ne prend fin
// qu'après l'appel à commit ou rollback

// commit pour la valider
connection.commit( );

// rollback pour la rejeter
connection.rollback( );
```

Chargement dynamique

Lors de l'exécution d'un programme Java,
les classes ne sont pas toutes chargées en mémoire .

Les classes sont chargées lorsque c'est nécessaire :
- quand la construction d'une instance(directe ou indirecte) est demandée
- quand une méthode de la classe doit être utilisée
- quand la méthode class.forName("nomDeLaClasseACharger") est appelée

La méthode **Class.forName** permet de forcer le chargement dynamique de classe

Bloc static

La définition d'une classe peut contenir des blocs statics.

```
public class C1{
private String s;
private static Date dateChargement;
static {
// nous sommes à l'intérieur d'un bloc static
//    (ils sont déclarés au même niveau que les membres de la classe)
System.out.println("Chargement de la classe C1 en cours");
// on peut dans un bloc static initialiser des données membres statics par exemple
dateChargement=new Date( );
// on peut aussi appeler des méthodes d'autres classes
C2.prevenirQueC1EstChargee( );
// en fait, on peut faire ce que l'on veut
}
// après, la définition de la classe continue
public C1(String nom){…}
...
```

LES BLOCS STATICS SONT EXECUTES UNE FOIS AU MOMENT DU CHARGEMENT DE LA CLASSE

$<chargedyn>

Chargement dynamique des drivers jdbc

La méthode **Class.forName** permet le chargement dynamique de classe
Les blocs statics de la classe concernée sont donc exécutés.

Revenons à nos drivers jdbc ...

Toutes les classes de drivers ont un bloc static qui fait les 2 opérations suivantes:
- création d'une instance de la classe en chargement
- appel d'une méthode de la classe DriverManager
prenant en paramètre cette instance
et ayant pour rôle de l'insérer dans la liste des drivers disponibles.

Ainsi, au moment de l'appel de getConnection de DriverManager,
le driver est connu et la connexion au SGBD peut être effectué.

Class.forName et import

Pourquoi Class.forName("com.mysql.jdbc.Driver");
et pas import com.mysql.jdbc.Driver; ?

import <nomDeClasse> ;
C'est un alias évitant de mettre le nom complet des classes dans le programme :
cela ne garantie en rien que la classe sera chargée

import com.mysql.jdbc.Driver; // insuffisant ici car on veut que la classe soit chargée pour être enregistrée auprès du DriverManager avant l'appel de getConnection.

Autre différence :
- **import** nécessite de mettre le nom de la classe "en dur" dans le programme,
or la classe du Driver est souvent mise dans un fichier de configuration

String className = lireFichier("conf.xml");
Class.forName(className);
Cela permet d'utiliser le même programme avec des SGBD différents sans aucune modification

Utilisation d'un fichier de configuration

Bd.properties

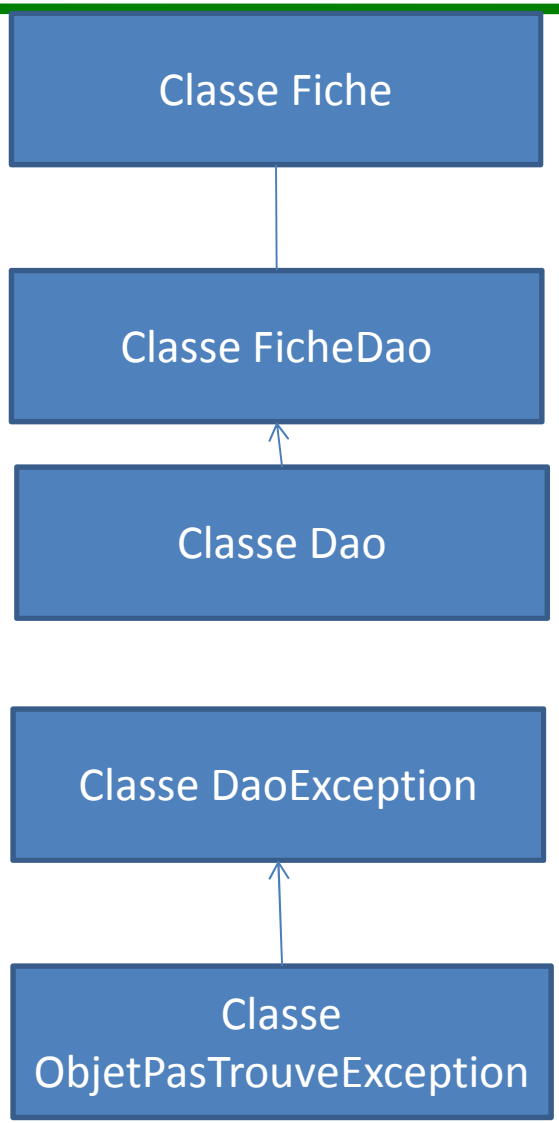
```
login=monnom
password=monpass
url=jdbc:sqlite:testsqlite.db
driver=org.sqlite.JDBC
```

```
//InputStream fis =new FileInputStream("bd.properties");
InputStream fis =this.getClass( ).getResourceAsStream("/"+fileName);

Properties properties = new Properties( );
properties.load(fis);

System.out.println(properties.getProperty("driver")); // org.sqlite.JDBC
```

La classe BD

```
/** une classe pour obtenir facilement des connexions */
public class BD {

private  Properties properties = new Properties();


public BD(String fileName) throws IOException, ClassNotFoundException {
        try (InputStream input = BD.class.getResourceAsStream("/"+fileName))
        {
            properties.load(input);
        }
        Class.forName(properties.getProperty("driver"));
}

/** rend une connection à la base de données*/
public   Connection getConnection( ) throws SQLException{
    return DriverManager.getConnection(
            properties.getProperty("url"),
            properties.getProperty("login"),
            properties.getProperty("password"));
            }
}
```

Exemple de mapping objet-relationnel



Login, nom, prenom

Opérations SQL

Exceptions lancées par les méthodes de FicheDao

Une sous classe de DaoException

La classe Fiche

Un aperçu de la classe Fiche …

```
public class Fiche {

private String login,prenom,nom;
……
Fiche(String login, String prenom, String nom) {
        this.login = login;
        this.prenom = prenom;
        this.nom = nom;
        }

public String getPrenom( ) …
public void setPrenom(String prenom) …

public String getNom( ) …
public void setNom(String nom) …

public String getLogin( ) …

public String toString( ) …
}
```

La classe Fiche et la classe FicheDao

Un objet FicheDao est utilisé dans la classe Fiche pour l'accès aux bases de données

DAO = Data Access Object

```
public class Fiche {

private static FicheDao dao;

public static void setDao(FicheDAO dao) {
        Fiche.dao=dao;
}
private String login,prenom,nom;

public String getPrenom() {
        return this.prenom;
}
public void setPrenom(String prenom) throws DAOException {
        this.prenom = prenom;
        dao.mettreAJour(this);
}
……
```

Classes d'exception

Les deux classes d'exceptions utilisées

```
public class DaoException extends Exception {

  public DaoException(String message) {
    super(message);
  }
  public DaoException(Throwable cause) {
    super(cause);
  }
}
public class ObjetPasTrouveException extends DaoException {

  public ObjetPasTrouveException(String message){
    super(message);
  }
  public ObjetPasTrouveException(Throwable cause) {
    super(cause);
  }
}
```

La classe Fiche : autres méthodes d'instances

Continuons et terminons avec la classe Fiche …

```
public String getNom() {
   return nom;
}
public void setNom(String nom) throws DaoException {
   this.nom = nom;
   dao.mettreAJour(this);
}
public String getLogin() {
   return login;
}
public String toString() {
   return "Fiche [login=" + login + ", prenom=" + prenom + ", nom=" + nom + "]";
}
```

Un exemple d'utilisation

Un exemple d'utilisation …

```
BD bd = new BD("bd.properties");

FicheDao dao = new FicheDao(bd);
Fiche.setDao(dao);

Fiche f1= dao.trouver("durand");
Fiche f2= dao.creer("dupond","Pierre","Dupond");

f2.setNom("Dupont");

System.out.println(f1.getPrenom());
System.out.println(f2);

dao.supprimer(f1);

f1=dao.trouver("durand"); // erreur
```

Dao : des méthodes générales

Savoir si une fiche est dans la base …

```
public abstract class Dao<E>{

private BD bd;

public  Dao(BD bd) {
      this.bd=bd;
       }

protected Connection getConnection() throws SQLException{
            return this.bd.getConnection();
}

protected  boolean isTableExiste(String nomTable) throws DaoException{…}

public  abstract E trouver(String id) throws DaoException;

public  boolean isExisteDansLaBase(String id) throws DaoException{…}

}
```

Dao : isExisteDansLaBase

Savoir si une fiche est dans la base …

```
/** teste si l'objet d'identifiant donné existe en base
 * @param id l'identifiant
 * @return true si l'objet existe en base, faux sinon
 * @throws DaoException
 */
public  boolean isExisteDansLaBase(String login) throws DaoException{
    try {
       trouver(login);
       return true;
    }
    catch (ObjetInconnuException exp)  {
       return false;
      }
  }
```

Dao : isTableExiste

Savoir si une table existe …

```
/**
 * teste si une table existe dans la base de données
 * @param nomTable
 * @return vrai si la table existe, faux sinon
 * @throws DaoException  si une erreur apparait au cours de ce test
 */
private  boolean isTableExiste(String nomTable) throws DaoException{
     try  (Connection connection = bd.getConnection();
           Statement statement = connection.createStatement()) {
        try {
          statement.executeQuery("SELECT * from "+nomTable);
          return true;
        }
        catch (SQLException exp) {return false;}
     }
     catch (SQLException exp) {throw new DaoException(exp);}

  }
```

FicheDao

```
public class FicheDao extends Dao<Fiche>{

   public  FicheDao(BD bd) throws DaoException{...}

   public  Fiche creer(String login,String nom,String prenom)throws  DaoException{...}

   @Override
   public  Fiche trouver(String login) throws DaoException{...}

   public void mettreAJour(Fiche fiche) throws DaoException{...}

   public void supprimer(Fiche fiche) throws DaoException{...}

}
```

FicheDao : le constructeur

Le constructeur de FicheDao… qui crée la table si nécessaire

```
public  FicheDao(BD bd) throws DaoException{

    super(bd);

    if (! isTableExiste("fiche")){
       try  (Connection connection = this.getConnection();
            Statement statement = connection.createStatement( )) {
         statement.executeUpdate("CREATE TABLE fiche " +
              "(login VARCHAR(25) PRIMARY KEY,nom VARCHAR(25),prenom VARCHAR(25))");
       }
       catch (SQLException exp) {throw new DaoException(exp);}
    }
}
```

FicheDao : creer

Création d'une fiche en base de données

```
public  Fiche creer(String login,String nom,String prenom)throws  DaoException{
  if (isExisteDansLaBase(login))
      throw new DaoException("Fiche déjà existante: "+login);
    else {
      try (Connection connection = this.getConnection();
           PreparedStatement statement = connection.prepareStatement(
             "INSERT INTO fiche (login,nom,prenom) VALUES (?,?,?)" )){
        statement.setString(1, login);
        statement.setString(2, nom);
        statement.setString(3, prenom);

        statement.executeUpdate();
      }

      catch (SQLException exp) {throw new DaoException(exp);}

      return trouver(login);

  }
}
```

FicheDao : trouver

Retrouver une fiche à partir de sa clef primaire

```
public  Fiche trouver(String login) throws DaoException{
   try (Connection connection = this.getConnection();
         PreparedStatement statement =
            connection.prepareStatement("SELECT nom,prenom FROM fiche WHERE login=?")){
         statement.setString(1,login);

      ResultSet rs= statement.executeQuery( );
      if (!rs.next( ))
         throw new ObjetInconnuException("Fiche inexistante : "+login);
      else return new Fiche(login,rs.getString(1),rs.getString(2));
   }
   catch (SQLException exp) {throw new DaoException(exp);}
}
```

FicheDao : mettreAJour

Mettre à jour une fiche dans la base …

```
public void mettreAJour(Fiche fiche) throws DaoException{
  try (Connection connection = this.getConnection();
       PreparedStatement statement = connection.prepareStatement(
            "UPDATE fiche SET nom=? ,prenom=?  WHERE login=?")){
    statement.setString(1,fiche.getNom());
    statement.setString(2, fiche.getPrenom());
    statement.setString(3, fiche.getLogin());

    statement.executeUpdate();
  }
  catch (SQLException exp) {throw new DaoException(exp);}

}
```

FicheDao : supprimer

Et pour terminer, suppression d'une fiche en base …

```
public void supprimer(Fiche fiche) throws DaoException{
  try  (Connection connection = this.getConnection();
        PreparedStatement statement = connection.prepareStatement(
            "DELETE FROM fiche  WHERE login=?")) {

    statement.setString(1, fiche.getLogin());

    statement.executeUpdate();
  }

  catch (SQLException exp) {throw new DaoException(exp);}

}
```

Un autre exemple : la classe Compte

Nous allons définir une classe Compte avec les fonctionnalités habituelles (ajout d'une opération, consultation du solde...) et sauvegarde en base de données

Nous allons voir :

1. Une exemple d'utilisation de la classe Compte

2. La base de données utilisée

3. La définition de la classe Compte

3. La définition de la classe CompteDao

Utilisation de la classe CompteBD

```
public static void main(String[ ] args) throws Exception {
    BD bd = new BD("bd.properties");
    CompteDao dao = new CompteDao(bd);
    Compte.setDao(dao);

    Compte c1=null;
    try{c1=dao.trouver("1001");}                    }
    catch(DaoException exp) {
          c1=dao.creer("1001","Dupond");
    }
    c1.addOperation(120);
    c1.setDecouvertAutorise(534);
    System.out.println(c1.getNomTitulaire( )+" "+c1.getSolde( )+
                " "+c1.getDecouvertAutorise( ));
    }}
```

La base de données associée

2 tables

compte	**num**	**INTEGER(16)**
	nom	**VARCHAR(80)**
	solde	**INTEGER(16)**
	decouvert	**INTEGER(16)**
operation	**num**	**INTEGER(16)**
	montant	**INTEGER(16)**

Les montants sont enregistrés en centimes d'euros.

La classe Compte : les attributs

Deux attributs d'instance : numero, nomTitulaire
Le solde, le découvert autorisé et l'historique
sont mémorisés dans la base de données

```
/** le titulaire de ce compte*/
private String numero;

/** le titulaire de ce compte*/
private String nomTitulaire;
```

la classe Compte

```
private String numeroCompte;
private String nomTitulaire;

private static CompteDao dao;

public static void setDao(CompteDao dao) {Compte.dao=dao;}

/** creation d'un Compte */
Compte(String numeroCompte,String nomTitulaire){
        this.numeroCompte=numeroCompte;
        this.nomTitulaire=nomTitulaire;
        }

/** enregistre une opération de 'montant' Euros sur ce compte
 * @param montant le montant en euros de l'opération */
public void addOperation(double montant)throws DaoException{
        dao.addOperation(numeroCompte, montant);
        this.setSolde(this.getSolde( )+montant);
        }
```

Les méthodes d'instance de la classe Compte

```
public void setDecouvertAutorise(double decouvertAutorise)throws DaoException{
      dao.setDecouvertAutorise(numeroCompte, decouvertAutorise);
}

private void setSolde(double montant)throws DaoException{
      dao.setSolde(numeroCompte, montant);
}

public double getSolde( ) throws  DaoException{
      return dao.getSolde(numeroCompte);
}

public double getDecouvertAutorise( ) throws  DaoException{
      return dao.getDecouvertAutorise(numeroCompte);
}

public String getHistorique( ) throws  DaoException{
      return dao.getHistorique(numeroCompte);    }
```

Les méthodes d'instance de la classe Compte

```
public boolean isSoldeInsuffisant( ) throws DaoException {
      return this.getSolde( ) <   - this.getDecouvertAutorise();
}

public String getNumeroCompte( ) {
      return this.numeroCompte;
}
public String getNomTitulaire( ) {
      return this.nomTitulaire;
}

public String toString( ){
      return("Compte de "+this.numeroCompte);
}
```

La classe CompteDao

public class CompteDao extends Dao<Compte>{

public CompteDao(BD bd){...}

public Compte creer(String numeroCompte,String nomTitulaire) throws DaoException{...}

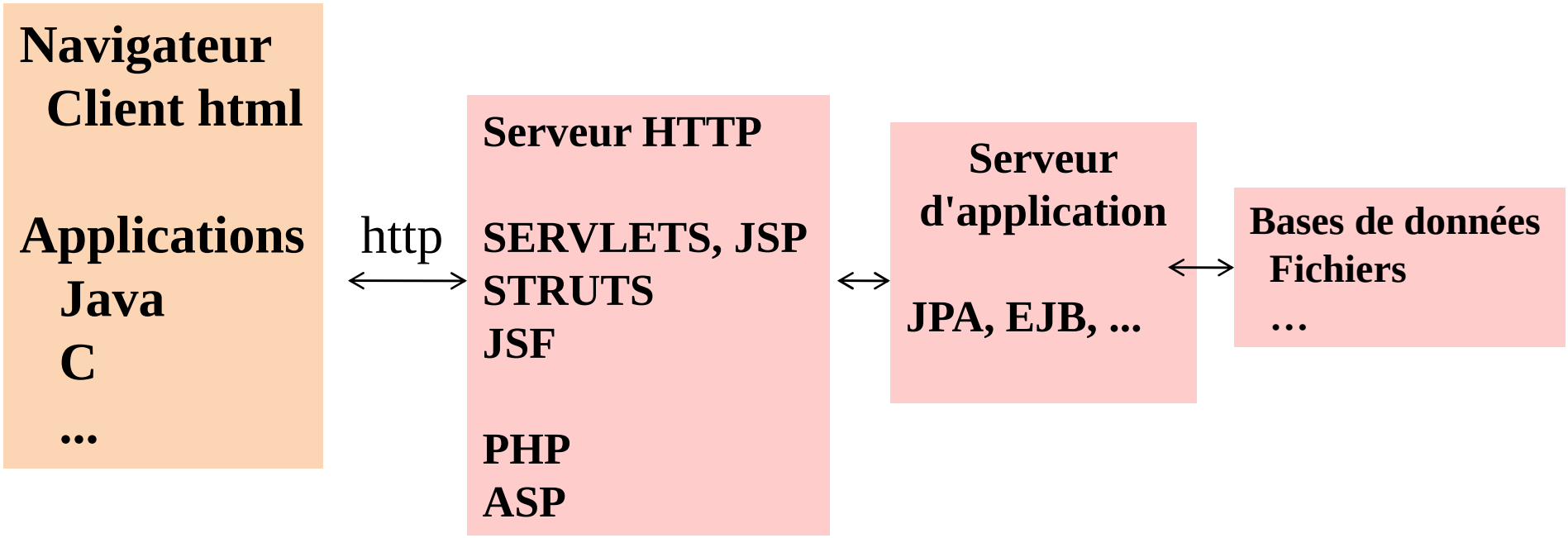
@Override
public Compte trouver(String numeroCompte) throws DaoException{...}

public void addOperation(String numeroCompte,double montant) throws DaoException{...}
public void setDecouvertAutorise(String numeroCompte,double decouvertAutorise)
throws DaoException{...}
public void setSolde(String numeroCompte,double solde) throws DaoException{...}

public double getSolde(String numeroCompte) throws DaoException{...}
public double getDecouvertAutorise(String numeroCompte) throws DaoException{...}

public String getHistorique(String numeroCompte) throws DaoException{...}}

Applications WEB



http et conteneur de servlet

servlet

instance d'une classe de Servlet
(HttpServlet pour le protocole http)

conteneur de servlet

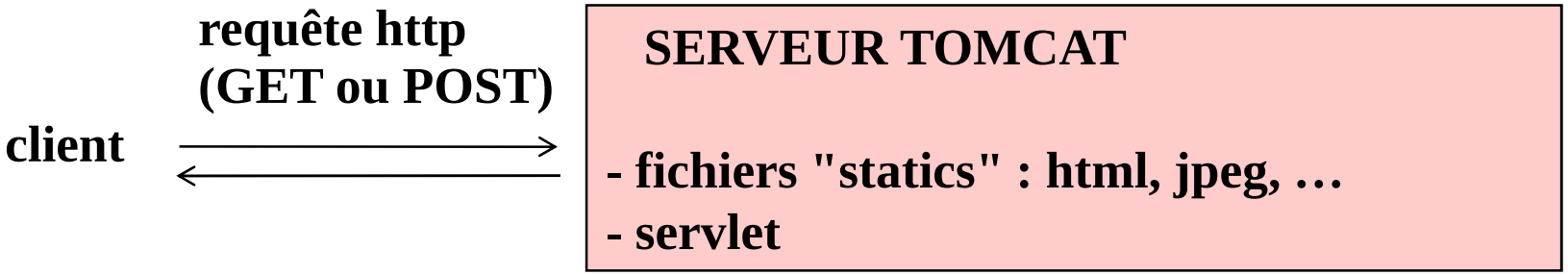
Un container de servlet a pour rôle
de permettre l'invocation de servlets par un client.
Il présente aux servlets la requete http et la réponse à cette requête sous forme d'objets Java

- réception d'une requete http
- détermination de la servlet à appeler
- construction des objets request et response
- appel de la méthode appropriée de la servlet
- renvoie de la réponse http

Exemple de conteneur de servlet : tomcat, weblogic, websphere, …

SERVEUR TOMCAT

- Les servlets Java sont des objets Java coté serveurs pouvant être invoquées par un client grâce au protocole Http
- Tomcat est un serveur http permettant l'invocation de servlet.



Le résultat d'une requête invoquant une servlet peut être :
- un flux de caractères (exemple : une page html)
- un flux d'octets (nombres, tableaux, objets sérialisés …)

Serveur Tomcat et contextes

- Plusieurs applications indépendantes peuvent être exécutés sur un même serveur : elles utilisent chacune un contexte spécifique
- Tomcat permet de gérer plusieurs contextes :
  - »un par application
  - »A l'intérieur d'un contexte, on partage des données communes
  - »Pas de communication directe entre les contextes : tout se passe comme s'il y avait une machine virtuelle par contexte

- Le contexte est précisé dans l'url de la requête :
  - •http://descartes.math-info.univ-paris5.fr:8080/plenadis/index.html
  - •http://descartes.math-info.univ-paris5.fr:8080/etu002/servlet/essai.HelloWorld

- Chaque contexte possède son fichier de configuration (web.xml)

La classe HttpServlet

Les classes de servlet sont des classes dérivées de javax.servlet.http.HttpServlet

Deux méthodes de javax.servlet.http.HttpServlet traitant les méthodes GET et POST:

public void doGet(HttpServletRequest req,HttpServletResponse res)
public void doPost(HttpServletRequest req,HttpServletResponse res)
Par défaut, ces deux méthodes telles qu'elles sont définies dans la classe HttpServlet renvoient une page indiquant au client que le service n'est pas traité.

GET	**les paramètres font partie de l'url**
POST	**les paramètres sont transmis dans le corps de la requête HTTP (taille des paramètres quelconques)**

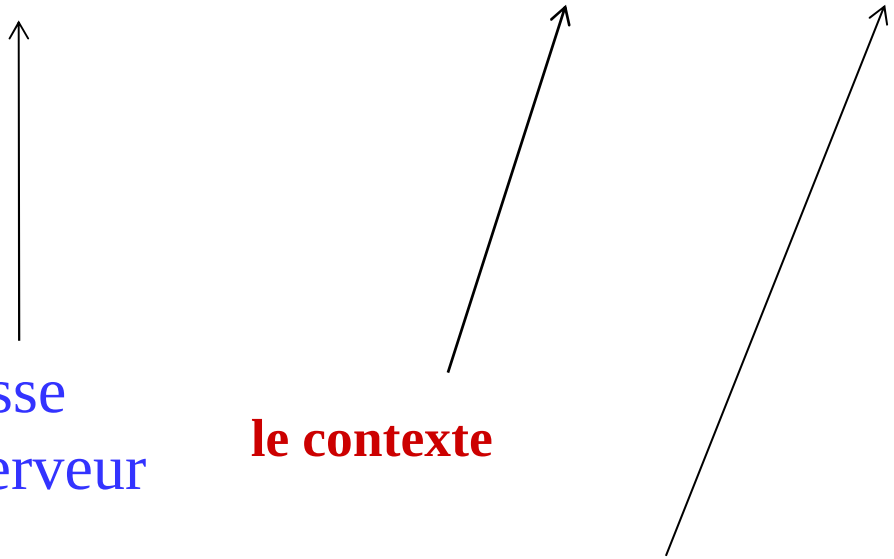
Une première Servlet qui renvoie du html

```
package essai;
import java.io.*;import javax.servlet.*;import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {
public void doGet(HttpServletRequest req,HttpServletResponse res)
                    throws ServletException, IOException {
    // type mime du résultat retourné
    res.setContentType("text/html");
    // récupère le flux de sortie  vers le client
    PrintWriter out = res.getWriter( );

    out.println("<html>");
    out.println("<head><title>Hello World</title></ head >");
    out.println("<body>");
    out.println("<font size=\"+3\">Hello World<br/></font>");
    out.println("</ body >");
    out.println("</ html >");
    }}
```

Cette servlet renvoie une page HTML

Appel de la servlet HelloWorld

http://192.168.0.200:8080/moncontexte/go



adresse
du serveur

le contexte

pour indiquer au serveur
d'utiliser la servlet HelloWorld

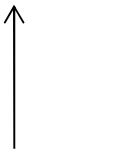
Une Servlet avec paramètres

```
package essai;
import java.io.*;import javax.servlet.*;import javax.servlet.http.*;

public class HelloWorld2 extends HttpServlet {
public void doGet(HttpServletRequest req,HttpServletResponse res)
  throws ServletException, IOException , NumberFormatException{
      res.setContentType("text/html");

      PrintWriter out = res.getWriter( );

      out.println("<html>");
      out.println("<head><title>Hello World</title></head>");
      out.println("<body>");
      String size = req.getParameter("taille");
      int n = Integer.parseInt(req.getParameter("repeter"));
      for (int i=0;i<n;i++)
            out.println("<font size="+size+">Hello World<br/></font>");
      out.println("</body>");
      out.println("</html>");
      }
```

Appel de la servlet HelloWorld2

url d'appel de la servlet

http://192.168.0.200:8080/moncontexte/go?taille=2&repeter=3



paramètres

appel de la servlet à partir d'un formulaire

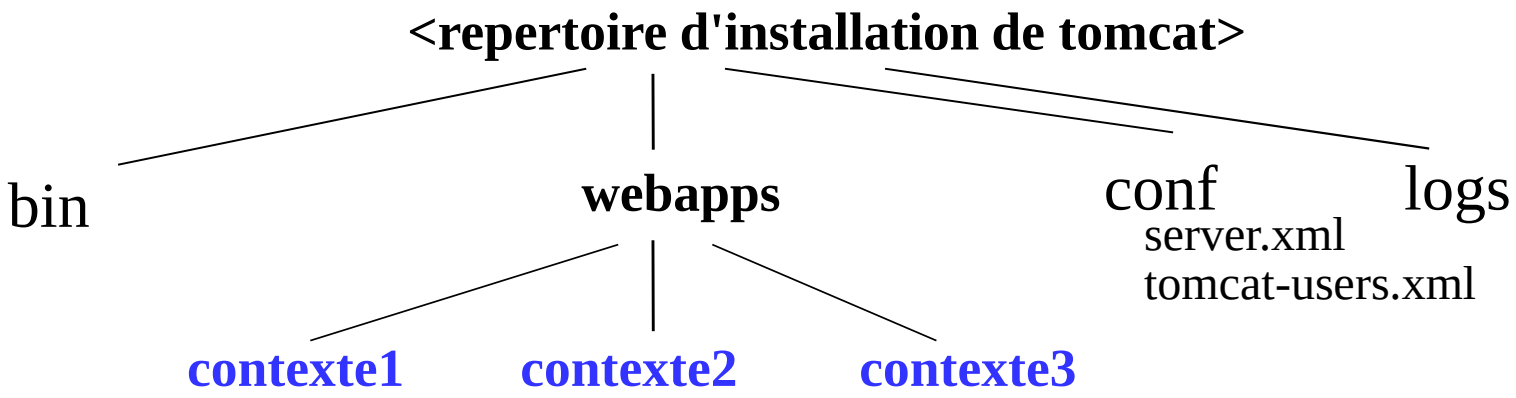
```
<html>
<head><title>essai</title></head>
<body>
  <form action="/ moncontexte /go" method="get">
     <input type="text" name="taille">taille
     <input type="text" name="repeter">nombre de lignes
     <input type="submit" name="OK">
  </form>
</body>
</html>
```

résultat

```
<html>
<head><title>Hello World</title></head>
<body>
<font size=2>Hello World<br/></ font>
<font size=2>Hello World<br/></ font>
<font size=2>Hello World<br/></ font>
</body>
</html>
```

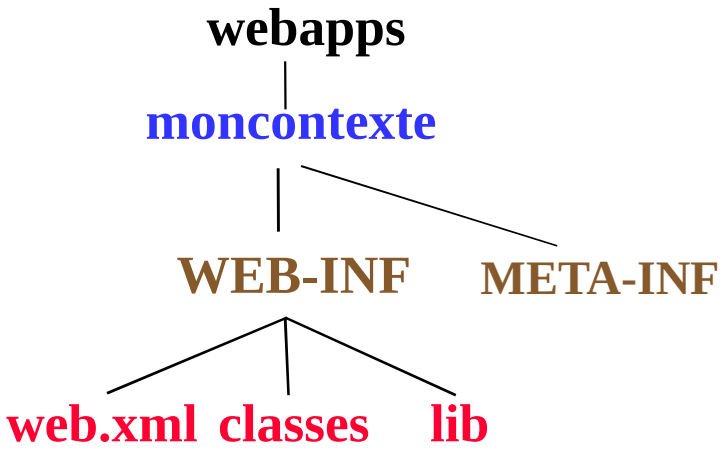
page html générée par la servlet

Tomcat : organisation des fichiers



bin: commandes de lancement et d'arrêt du serveur
webapps : à chaque contexte correspond un répertoire de webapps
conf : fichiers de configuration
logs : fichiers de logs

Tomcat : contexte



le répertoire de base du contexte

- les pages html

- le répertoire de base des classes d'applets

WEB-INF (répertoire privé)

web.xml // le fichier de configuration du contexte

classes // répertoire de base des classes utilisées par le moteur de servlet
// (servlets et classes utilisées par les servlets)

lib // répertoire de base des fichiers jar utilisés par le moteur de servlet
// (servlets et classes utilisées par les servlets)

META-INF (répertoire privé)

Exceptés les répertoires WEB-INF et META-INF,
les fichiers d'un contexte sont accessibles publiquement.

Exemple de contenu du répertoire du contexte

Ceci est le contenu du répertoire d'un contexte vide (pas de fichier excepté web.xml)

webapps
moncontexte
WEB-INF
web.xml

Ceci est le contenu du répertoire du contexte avec la servlet essai.HelloWorld

webapps
moncontexte
WEB-INF
web.xml
classes
essai
HelloWorld.class

Configuration du contexte : web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                    http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
            version="3.1"
        metadata-complete="true">
<servlet>
      <servlet-name>hello</servlet-name>
      <servlet-class>essai.HelloWorld</servlet-class>
</servlet>
 <!-- pour indiquer que /go désigne l'appel  à la servlet essai.HelloWorld→
<servlet-mapping>
      <servlet-name>hello</servlet-name>
      <url-pattern>/go/*</url-pattern>
</servlet-mapping>
</web-app>
```

L'appel à la servlet essai.HelloWorld se fait alors par :
http://urlDuServeur:8080/moncontexte/go

La prise en compte de la modification de web.xml nécessite le redémarrage du contexte

Pour tester l'exemple HelloWorld

1. Compiler la classe essai.HelloWorld

2. Copier le fichier HelloWorld.class à l'emplacement indiqué précédemment

3. Appeler, depuis un navigateur, l'url suivante :
http://urlDuServeur:8080/moncontexte/go

ATTENTION :
Si vous modifiez votre classe, après avoir déposé la nouvelle version dans WEB-INF/classes, vous devez, pour que la modification soit prise en compte :
- soit redémarrer le serveur (si vous êtes autorisé à le faire)
- soit relancer le contexte en utilisant le manager
(url du manager : http://urlDuServeur:8080/manager/html)

Accès au manager

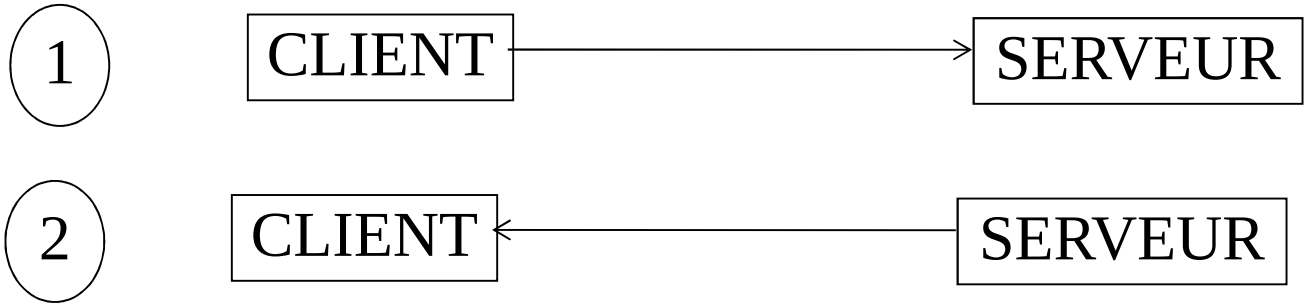
Pour avoir accès au manager,
il faut être inscrit dans le fichier tomcat-users.xml
avec comme rôle manager-gui.

<tomcat>/conf/tomcat-users.xml

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users      xmlns="http://tomcat.apache.org/xml"
                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-users.xsd"
                   version="1.0">
  <role rolename="manager-gui"/>
  <user username="admin" password= "secret" roles="manager-gui"/>
</tomcat-users>
```

Communication client serveur avec un client Java

Requête http



Le client
-envoie une requête au serveur

-récupère la réponse du serveur

Le serveur

-reçoit la requête du client
-renvoit la réponse au client

Communication client-serveur : coté client
Methode GET

```
String protocol = "http", hostname="localhost:8080", contexte="pary",path="essai";

String params = "valeur=10&nombre=14"

String urlString = protocol+"://"+hostname+"/"+contexte+"/"+path+"?"+params;

// création d'un objet URL
URL url = new URL(urlString);

// création d'une URLConnection
URLConnection urlc = url.openConnection( );

// connection au serveur
urlc.connect( );

// récupération du flux d'entrée en provenance du serveur
InputStream is=  urlc.getInputStream( );

// puis lecture à partir du flux d'entrée
// de la réponse du serveur
```

Communication client-serveur : coté client
Méthode POST

```
String protocol = "http", hostname="localhost:8080", contexte="pary",path="essai";
// pas de paramètres dans l'url pour les requêtes POST
String urlString = protocol+"://"+hostname+"/"+contexte+"/"+path;
// création d'un objet URL
URL url = new URL(urlString);
// création d'une URLConnection
URLConnection urlc = url.openConnection( );

// paramétrage de la connexion
urlc.setDoOutput(true); // false pour GET (defaut), true pour POST
// connection au serveur
urlc.connect( );
// récupération du flux de sortie
OutputStream os = urlc.getOutputStream( );
// écrire dans le flux de sortie ce que l'on désire écrire vers l'url
……
// fermeture du flux de sortie
os.close( );// à ne pas oublier

// récupération du flux d'entrée en provenance du serveur
InputStream is=  urlc.getInputStream( );
// puis lecture à partir du flux d'entrée de la réponse du serveur
```

Communication client serveur
coté serveur : réception des données

Réception de couples paramètres-valeurs : (doGet ou doPost)

méthode getParameter de HttpServletRequest

```
String nom = request.getParameter("nom");
```

Réception de données : (doPost)

```
// Cas 1 : texte
BufferedReader reader = request.getReader( );
String ligne = reader.readLine( );
// Cas 2 : données
DataInputStream dis = new DataInputStream(request.getInputStream( ));
long n = dis.readLong( );
// Cas 3 : objets sérialisés
ObjectInputStream ois = new ObjectInputStream(request.getInputStream( ));
Object obj = ois.readObject( );
```

Communication client serveur
coté serveur : envoi de la réponse

Envoi d'une réponse :

méthode getWriter de HttpServletResponse

```
PrintWriter writer = response.getWriter( );
writer.println(Math.PI);
writer.close( );
```

méthode getOutputStream de HttpServletResponse

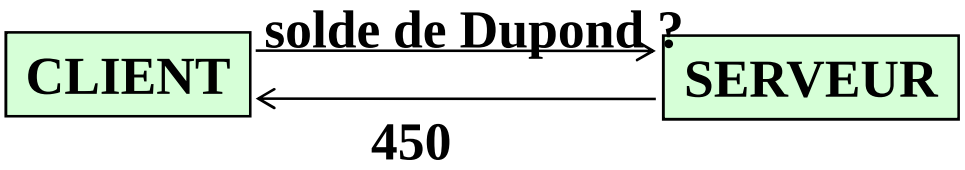
```
DataOutputStream dos = new DataOutputStream(response.getOutputStream( ));
dos.writeDouble(Math.PI);
dos.close( );
```

```
ObjectOutputStream oos = new ObjectOutputStream(response.getOutputStream( ));
oos.writeObject(new Date( ));
oos.close( );
```

Encodage des URL

```
/** Converts the properties list 'args' to a URL-encoded query string */
 private static String toEncodedString(Properties args) {
   StringBuffer buf = new StringBuffer( );
   Enumeration names = args.propertyNames( );
   while (names.hasMoreElements( )) {
     String name = (String) names.nextElement( );
     String value = args.getProperty(name);
     buf.append(URLEncoder.encode(name) + "=" + URLEncoder.encode(value));
     if (names.hasMoreElements( )) buf.append("&");
    }
   return buf.toString( );
 }
```

Exemple avec la classe Compte



Coté client : une méthode getSoldeCompte
qui fait appel à la servlet pour récupérer le solde du compte

Coté serveur : récupération du nom du titulaire et renvoie du solde

appel du client à la servlet(méthode GET)

```
private static double getSoldeFromServeur(URL urlServeur, String nomTitulaire)
        throws Exception{
  URL url = new URL(urlServeur,"/moncontexte/compte/?nom= "+nomTitulaire);

  URLConnection urlc = url.openConnection( );

  urlc.connect( );

  InputStream is =  urlc.getInputStream( );

  DataInputStream ois = new DataInputStream(is);
  return(ois.readDouble( ));
}
```

Méthode doGet

```
public void doGet(HttpServletRequest req,HttpServletResponse res) {
try {// 1 récupération de la valeur du paramètre
      String nomTitulaire = req.getParameter("nom");
      // 2 récupération du compte correspondant
      Compte compte = Compte.charger(nomTitulaire);
// 3 création du DataOuputStream pour la réponse au client
DataOutputStream dos = new DataOutputStream(res.getOutputStream( ));
// 4 envoie du solde du compte
dos.writeDouble(compte.getSolde( ));
   }
catch (Exception ex){res.setStatus(500);System.out.println(ex);}
}
```

Le serveur ne créé qu'une seule instance de chaque classe de Servlet

```
public class Compteur extends HttpServlet {
private int compteur=0;
public void doGet(HttpServletRequest req,HttpServletResponse res)
                throws ServletException, IOException {
    res.setContentType("text/html");

    PrintWriter out = res.getWriter( );

    out.println("<html>");
    out.println("<head><title>Compteur</title></head>");
    out.println("<body>");
    this.compteur++;
    out.println("Vous êtes le "+this.compteur+ "ème visiteur");
    out.println("</body>");
    out.println("</ html>");
    }
}
```

Le serveur crée un thread pour chaque requête

```
public class Compteur extends HttpServlet {
private int compteur=0;
private int nbAccesSimultane=0;
public void doGet(HttpServletRequest req,HttpServletResponse res)
                    throws ServletException, IOException {
      res.setContentType("text/html");
      this.compteur++;this.nbAccesSimultane++;
      PrintWriter out = res.getWriter( );
      out.println("<html>");
      out.println("<head><title>Compteur</title></head>");
      out.println("<body>");
      out.println("Vous êtes le "+this.compteur+ "ième visiteur");
      out.println("Vous êtes="+this.nbAccesSimultane
                        +" à utiliser en même temps cette servlet");
      out.println("</body>");
      out.println("</ html>");
      this.nbAccesSimultane--;
      }
}
```

Le serveur crée un thread pour chaque requête

Plusieurs threads sont susceptibles d'accéder et
de modifier en même temps un même objet !!

Les portions de code devant être sécurisées doivent être mises à l'intérieur d'un bloc synchronized.

(deux threads ne peuvent exécuter en même temps sur la même instance deux portions de code "synchronized".

```
// le moyen le plus sur de ne pas avoir de problème
// pas efficace et déconseillé en général
public synchronized void doGet(HttpServletRequest req,HttpServletResponse res)
        throws ServletException, IOException {
```

variables d'environnement CGI

Méthodes d'accès

SERVER_NAME	req.getServerName( )
SERVER_SOFTWARE	this.getServletContext( ).getServerInfo( )
SERVER_PROTOCOL	req.getProtocol( )
SERVER_PORT	req.getServerPort( )
REQUEST_METHOD	req.getMethod( )
PATH_INFO	req.getPathInfo( )
PATH_TRANSLATED	req.getPathTranslated( )
SCRIPT_NAME	req.getServletPath( )
DOCUMENT_ROOT	req.getRealPath( )
QUERY_STRING	req.getQueryString( )
REMOTE_HOST	req.getRemoteHost( )
REMOTE_ADDR	req.getRemoteAddr( )
AUTH_TYPE	req.getAuthType( )
REMOTE_USER	req.getRemoteUser( )
CONTENT_TYPE	req.getContentType( )
CONTENT_LENGTH	req.getContentLength( )
HTTP_ACCEPT	req.getHeader("Accept")
HTTP_USER_AGENT	req.getHeader("User-Agent")
HTTP_REFERER	req.getHeader("Referer")

récupérer l'url d'appel

```
public static String getUrlRacineContexte(HttpServletRequest req) {
   String scheme = req.getScheme( ); // http
   String serverName = req.getServerName( ); // rubis.ens.math-info.univ-paris5.fr
   int serverPort = req.getServerPort( ); // 8080
   String contextPath = req.getContextPath( ); // /monLogin
   return scheme + "://" + serverName + ":" + serverPort + contextPath;
  }

public static String getUrlCompleteAppel(HttpServletRequest req) {
      String servletPath = req.getServletPath( );   // /servlet/essai.MaServlet
      String pathInfo = req.getPathInfo( );        //   /a/b
      String queryString = req.getQueryString( );         //   d=789&e=32

      String url = getUrlRacineContexte(req)+servletPath;
      if (pathInfo != null) url += pathInfo;
      if (queryString != null) url += "?"+queryString;
      return url;// http://rubis.ens.math-info.univ-paris5.fr:8080/monLogin/servlet/essai.MaServlet/a/b?d=789&e=32
   }
```

Codes d'état HTTP

Pour positionner le statut,
la méthode de javax.servlet.httpservlet.HttpServletResponse:
public void setStatus(int sc)

SC_OK	**200**	**tout s'est passé normalement**
SC_NO_CONTENT	**204**	**tout est OK mais il n'y a pas de réponse à retourner**
SC_MOVED_PERMANENTLY	**301**	**le nouvel emplacement est donné par l'entête location**
SC_MOVED_EMPORARILY	**302**	**le nouvel emplacement est donné par l'entête location**
SC_UNAUTHORIZED	**401**	**accès à la page non autorisé**
SC_NOT_FOUND	**404**	**page non trouvée**
SC_INTERNAL_SERVER_ERROR	**500**	**Le serveur ne peut satisfaire la requête à cause d'un problème interne**
SC_NOT_IMPLEMENTED	**501**	**Le serveur ne supporte pas la fonctionnalité nécessaire pour satisfaire la requête**
SC_SERVICE_UNAVAILABLE	**502**	**Le serveur est temporairement indisponible**

Suivi de session

HTTP est un protocole sans état:
il faut trouver le moyen d'identifier un client

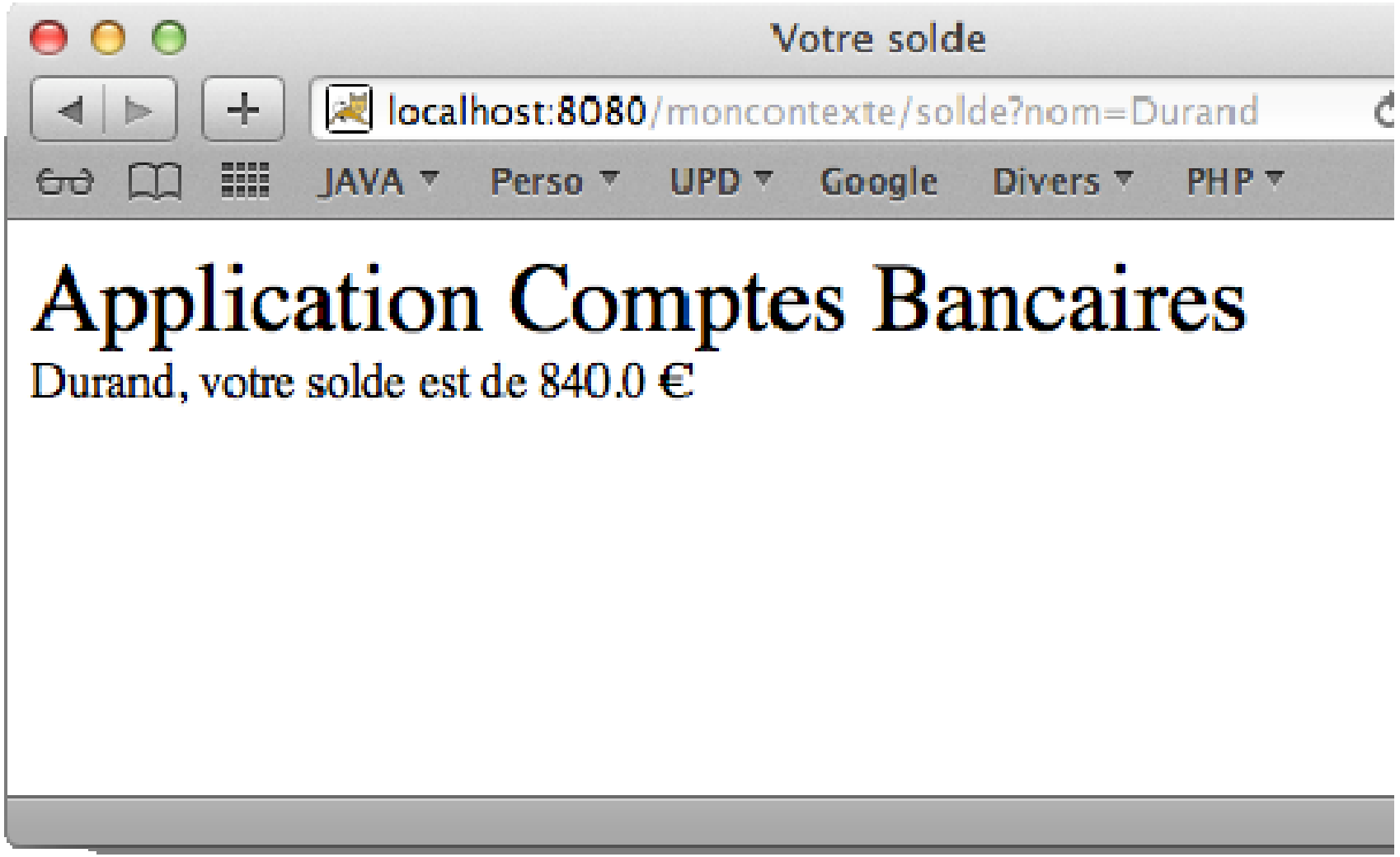
Des solutions pour le suivi de session sont :
- réécriture d'URL
- **cookies**

Les servlets permettent un suivi de session indépendant de la solution technique .

HttpSession

```
public class HttpServletRequest  extends GenericServlet{
……
/** rend la session de cette requete; si elle n'existe pas,
rend null ou cree une nouvelle session si 'create' */
public HttpSession getSession(boolean create)
}

public class HttpSession{
   /** associe 'value' à 'name' pour cette session*/
   public void putValue(String name,Object value){}
   /** rend la valeur associé à 'name' pour cette session*/
   public Object getValue(String name){}
   /** rend les noms ayant une valeur associée pour cette session*/
   public String[ ] getValueNames{}
   /** supprime la valeur associée à 'name' */
   public void removeValue(String name){}
   ..
}
```

Servlet de consultation du solde d'un compte



Servlet de consultation du solde d'un compte

http://localhost:8080/moncontexte/solde?nom=Dupond

Dupond, votre solde est de 480.0 €

http://localhost:8080/moncontexte/solde?nom=jean

Compte inexistant : jean

http://localhost:8080/moncontexte/solde

Aucun compte fourni

http://localhost:8080/moncontexte/solde?nom=Dupond
Erreur interne
java.sql.SQLException: [SQLITE_ERROR] SQL error or missing database (no such table: solde)
at org.sqlite.DB.newSQLException(DB.java:383)
…
at up5.mi.pary.jc.jdbc.compte.CompteBD.isCompteExisteDansLaBase(CompteBD.java:74)

Servlet de consultation du solde d'un compte
La méthode doGet

```
private CompteDAO dao;

public void doGet(HttpServletRequest req,HttpServletResponse res)
                                                    throws ServletException,
IOException {
                res.setContentType("text/html");
                PrintWriter out = res.getWriter( );

                out.println("<html>");
                out.println("<head><title>Votre solde</title></ head >");
                out.println("<body>");
                out.println("<font size=\"+3\">Comptes Bancaires<br/></font>");

                out.println(getReponse(req.getParameter("nom")));

                out.println("</ body >");
                out.println("</ html >");
```

Servlet de consultation du solde d'un compte
La méthode getReponse

```
public String getReponse(String nomTitulaire){
   if (nomTitulaire==null){
     return("Aucun compte fourni");
   }
   else {
     try {
        Compte compte= dao.trouver(nomTitulaire);
        return("Votre solde est :"+compte.getSolde());
     }
     catch (CompteInconnuException exp){
                    return("Compte inexistant : "+nomTitulaire);}
     catch (Exception exp){
        StringWriter sw = new StringWriter();
        exp.printStackTrace(new PrintWriter(sw));
        return("Erreur interne"+sw.toString());}
     }}
```

Servlet de consultation du solde d'un compte
La méthode init

```
public void init(ServletConfig config)throws ServletException{

  try {
    BD bd = new BD("../bd.properties"); // si bd.properties est dans WEB-INF

    dao = new CompteDAO(bd);

    Compte.setDAO(dao);

  }
  catch (Exception e) {throw new ServletException(e);}
}
```