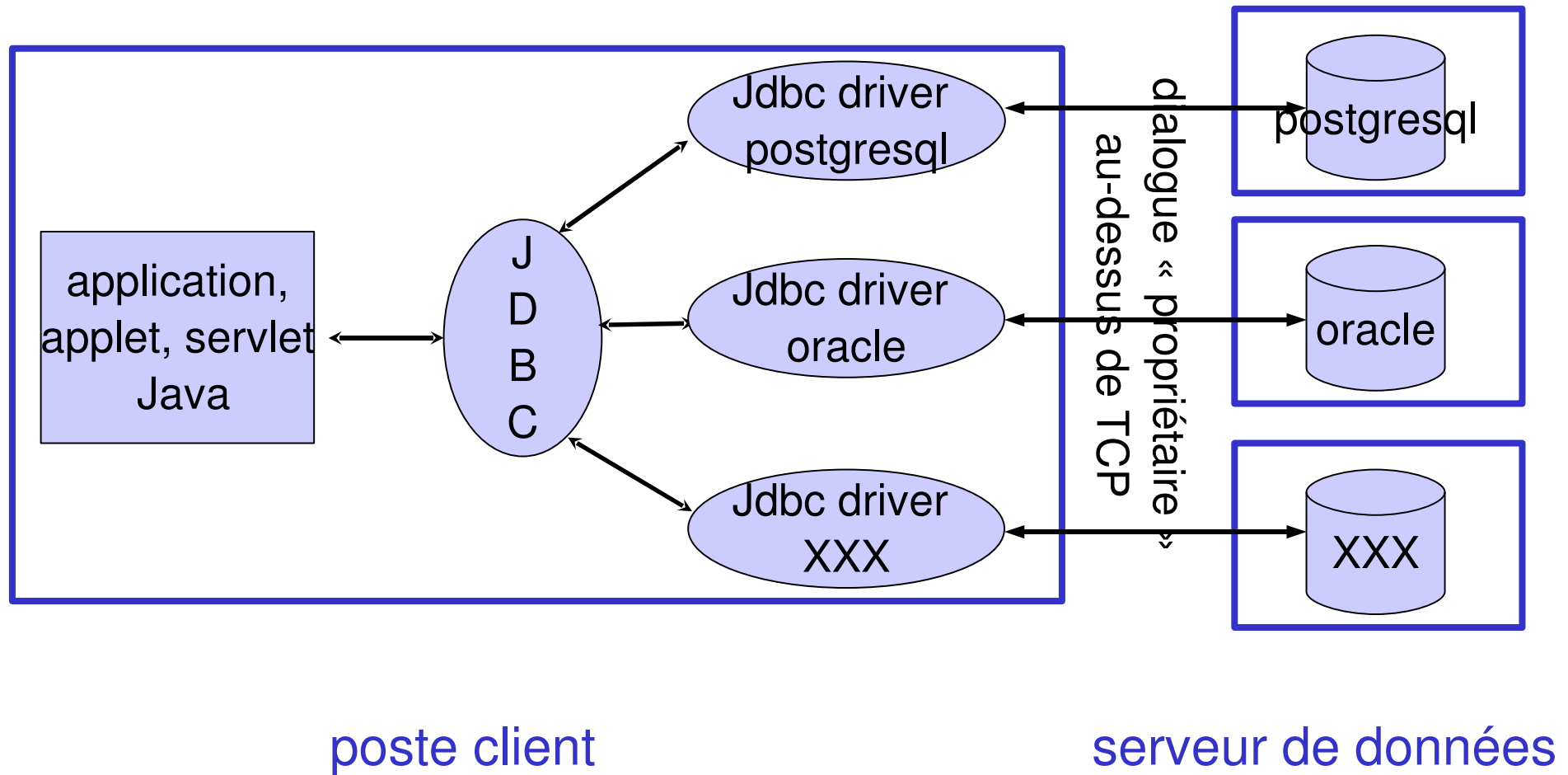


# ACCES AUX BASES DE DONNEES DEPUIS JAVA

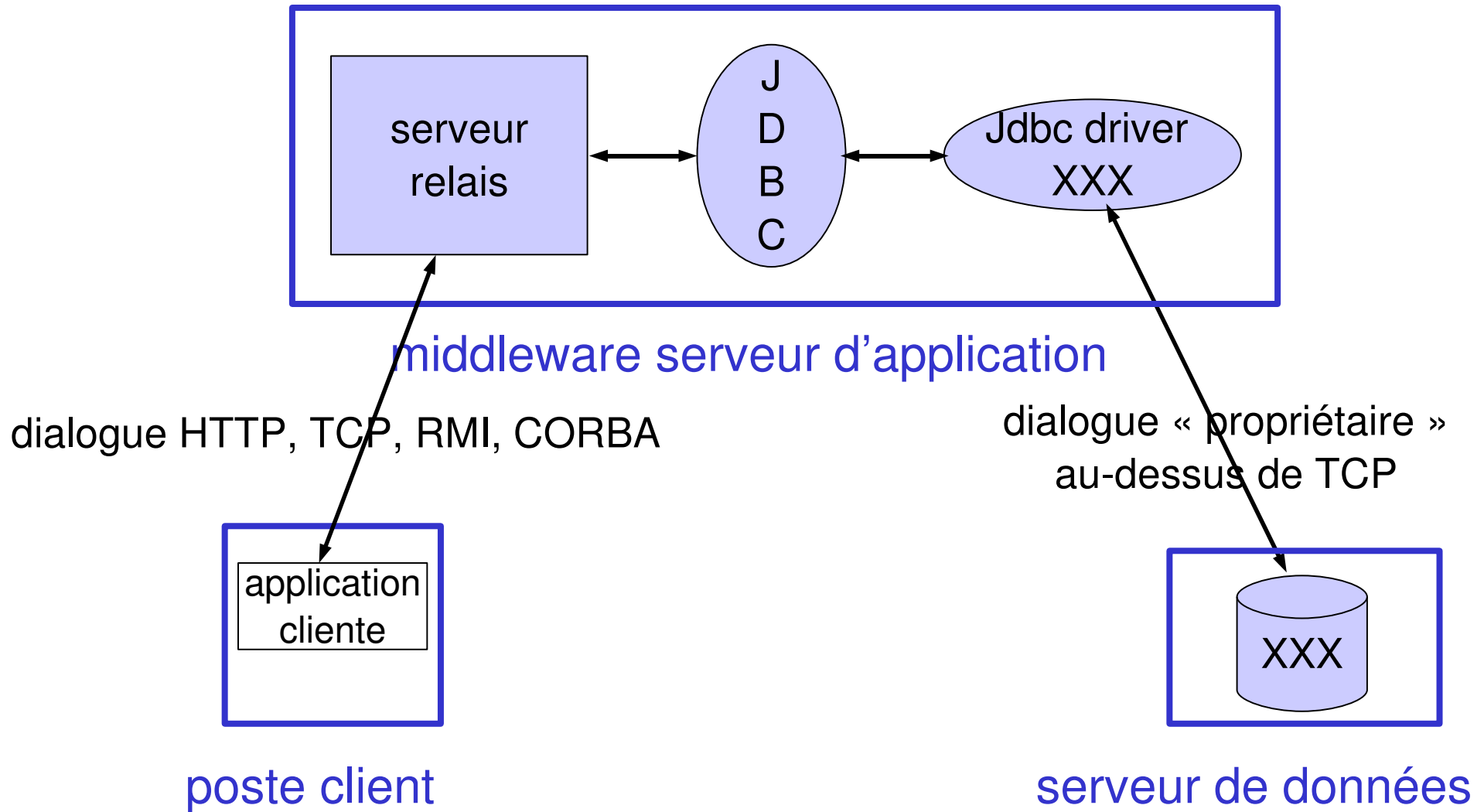
## JDBC : Java DataBase Connectivity

- permet
  - d'établir une connexion à un SGBD relationnel depuis une application, une applet ou une servlet JAVA
  - d'envoyer des requêtes SQL au SGBD
  - de traiter les résultats retournés par le SGBD
- grande portabilité vis-à-vis du SGBD visé, modulo la disponibilité et le chargement du driver spécifique
- le package `java.sql.*` fournit des interfaces et le driver du SGBD fournit les classes qui implémentent les interfaces et fournit donc toutes les classes utiles
- Il existe différents types (1,2,3,4) de drivers, voir :  
<http://developers.sun.com/product/jdbc/drivers>

# architecture 2-tiers



# architecture 3-tiers



# méthodologie générale de développement

- *Importer le package java.sql.\**
- Charger le driver adapté au SGBD visé (*a priori inutile pour un driver de type 4*)
- Créer (ou récupérer) une connexion (existante) au SGBD
- Créer une requête
- Exécuter la requête
- Traiter les résultats
- Fermer tous les objets

## principales méthodes

*ne pas oublier de spécifier à la machine virtuelle java l'emplacement du driver, soit par la redéfinition de la variable CLASSPATH soit lors du lancement de l'application par*  
*java -cp .:rép\_du\_driver monappli*

- charger le driver spécifique au SGBD visé (sauf pour un driver de type 4)
  - `Class.forName("org.postgresql.Driver")`
  - Une fois le driver chargé il s'enregistre auprès du `DriverManager`
- établir une connexion avec le SGBD
  - `Connection Co = DriverManager.getConnection(Base, User, Passwd)`
    - Base=jdbc:id\_sgbd://url\_sgbd:port\_sgbd/nom\_base
    - User=nom d'utilisateur
    - Passwd=mot de passe de User pour accéder à la BD nom\_base
  - Le `DriverManager` essaye d'établir la connexion avec tous les drivers qui ont été chargés.

- créer et exécuter une requête grâce à un objet de la classe **Statement** ou d'une de ses sous-classes **PreparedStatement** ou **CallableStatement**
  - requête simple
    - **Statement** req = Co.createStatement()
    - **ResultSet** Resu = req.executeQuery(requête sql)  
retourne le résultat de l'exécution de la **requête de consultation** passée en paramètre
    - **int** NbL = req.executeUpdate(requête sql)  
exécute la **requête de mise à jour** passée en paramètre et retourne le nombre de n\_uplets modifiés
  - requête compilée
    - **PreparedStatement** req=prepareStatement(requête sql avec ?)
    - req.setXXX(int, valeur\_type\_XXX) permet de donner une valeur à chaque ?
    - **int** NbL= req.executeUpdate() exécute la requête de mise à jour préparée dans req et retourne le nombre de n\_uplets modifiées ou 0 si la requête ne retourne rien
    - **ResultSet** Resu=req.executeQuery() retourne le résultat de l'exécution de la requête de consultation préparée dans req
  - procédure stockée
    - **CallableStatement** req=prepareCall(String sql)
    - **CallableStatement** est une sous-classe de **PreparedStatement** donc elle dispose de ses méthodes plus des méthodes supplémentaires pour gérer les paramètres données et résultats de la procédure à exécuter
  - les erreurs éventuelles dans les requêtes SQL seront détectées par le SGBD et remontées par le driver sous la forme d'une **SQLException**

- traiter les résultats
  - un objet de la classe **ResultSet** contient la table des résultats retournée par une requête de consultation
    - création à l'aide des méthodes **executeQuery()**
    - on parcourt chaque ligne d'un **ResultSet** grâce à la méthode **next()** qui retourne false lorsque le dernier enregistrement est dépassé (l'accès à la 1ère ligne nécessite un appel préalable à **next()**)
    - on accède aux données de chaque colonne par **getXXX**(nom de colonne) ou **getXXX**(n° de colonne) en adaptant XXX au type des données de la colonne visée
  - un objet de la classe **ResultSetMetaData** contient des informations sur le type et les propriétés d'une colonne d'un **ResultSet**
    - création par la méthode **getMetaData()** de la classe **ResultSet**
    - consultation des méta-données par les méthodes **getColumnCount()**, **getColumnType**(n° de colonne), etc...

## Un exemple

- Afficher le contenu de la table **personne** dans la base **etudinfo** sur le serveur Postgresql de **forge** dont la structure est :

Colonne	Type	Modificateurs
idpers	integer	not null default nextval('seqidpers'::regclass)
nom	character varying(20)	
prenom	character varying(20)	
age	integer	

Index :

« `personne_pkey` » PRIMARY KEY, btree (idpers)

- Et qui contient les données suivantes :

idpers	nom	prenom	age
405	Yop	LaCaille	15
426	test222	test222	11
416	ppaa	azertyuiop	777
....			

```

import java.sql.*;
/*
    Exemple d'accès à une table sur le SGBD Postgresql
    Syntaxe d'appel sur mon portable :
    java -cp /usr/share/java/postgresql-8.3-604.jdbc4.jar:. exempleJDBC
*/

public class exempleJDBC{

    public static void main(String arg[]) {
        try {
            // chargement du driver
            Class.forName("org.postgresql.Driver") ;
            System.out.println("driver org.postgresql.Driver chargé");
            try{
                // connexion à la base
                String base = "jdbc:postgresql://forge:5432/etudinfo";
                String user = "etudinfo";
                String passwd = "reso06";
                Connection co = DriverManager.getConnection(base,user,passwd);
                System.out.println("connexion à la base établie");
                // préparation de la requête
                String req="select * from personne";
                Statement sta = co.createStatement();
            }
        }
    }
}

```

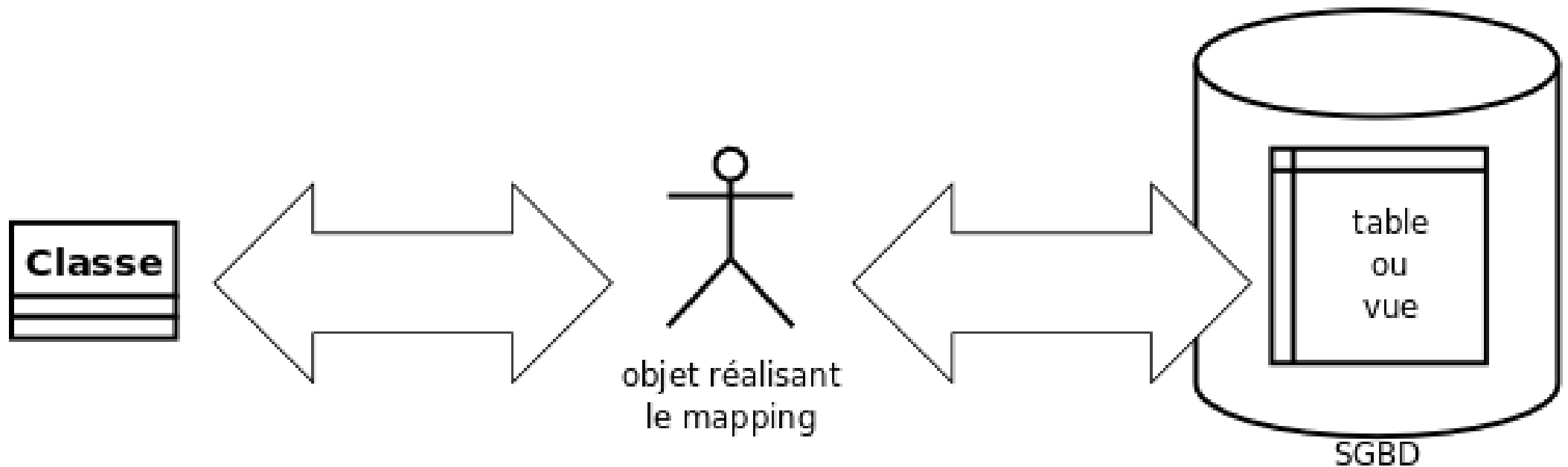
```

// exécution de la requête
ResultSet resu = sta.executeQuery(req);
// parcours du résultat
System.out.println("liste des personnes ");
while (resu.next()) {
    System.out.print(resu.getInt("idpers")+ " "+resu.getString("nom"));
    System.out.println(resu.getString("prenom")+
        " "+resu.getInt("age"));
}
// on referme tous les objets
resu.close();
sta.close();
co.close();
}
catch(SQLException sqle){
    System.out.println("problème avec la base de données");
    sqle.printStackTrace();
}
}
catch(ClassNotFoundException cnfe) {
    System.out.println("problème pour charger le driver org.postgresql.Driver");
    cnfe.printStackTrace();
}
}
}

```

## quelques notions de correspondance (mapping) objet-relationnel par l'exemple

- Buts :
  - mettre en correspondance des données contenues dans des tables (des relations) d'un SGBD avec des données contenues dans des objets d'une application.
  - avoir une séparation claire entre les traitements liés au SGBD (en SQL) et les traitements de l'application (en langage objet)
  - accroître la portabilité de l'application, son indépendance vis-à-vis du SGBD



## exemple de la gestion de la table personne

- La relation **personne(idpers, nom, prenom, age)** est représentée par la classe **Perso** qui dispose des mêmes attributs que ceux de la table du SGBD.
- La classe **Perso** possède différents constructeurs pour gérer différents cas :
  - on connaît l'identifiant idpers car l'objet est créé à partir d'un tuple qui est stocké dans la BD,
  - on ne connaît pas l'identifiant idpers car l'objet est créé à partir de données applicatives et n'existe pas encore dans la BD.
- Les attributs de la classe **Mapping** correspondent aux paramètres du SGBD auquel on souhaite accéder. Elle possède une méthode d'initialisation distincte du constructeur (réduit au minimum) pour gérer correctement les échecs éventuels de chargement de driver.
- Chaque traitement envisagé avec la BD est pris en charge par une méthode particulière.
- Toutes les exceptions éventuelles sont remontées à l'application via une nouvelle classe d'exception **MappingException**.
- Voir l'exemple dans fichiers Perso.java, Mapping.java, MappingException.java et exempleMapping.java.

## Pool de connexions

- Ouvrir (et refermer) une connexion particulière avec le SGBD pour chaque requête est « très » long.
- Un pool de connexions va ouvrir « à l'avance » un certain nombre de connexions, les maintenir ouvertes et les distribuer une à une pour chaque requête.
- Le pool va gérer cette allocation de connexions.

