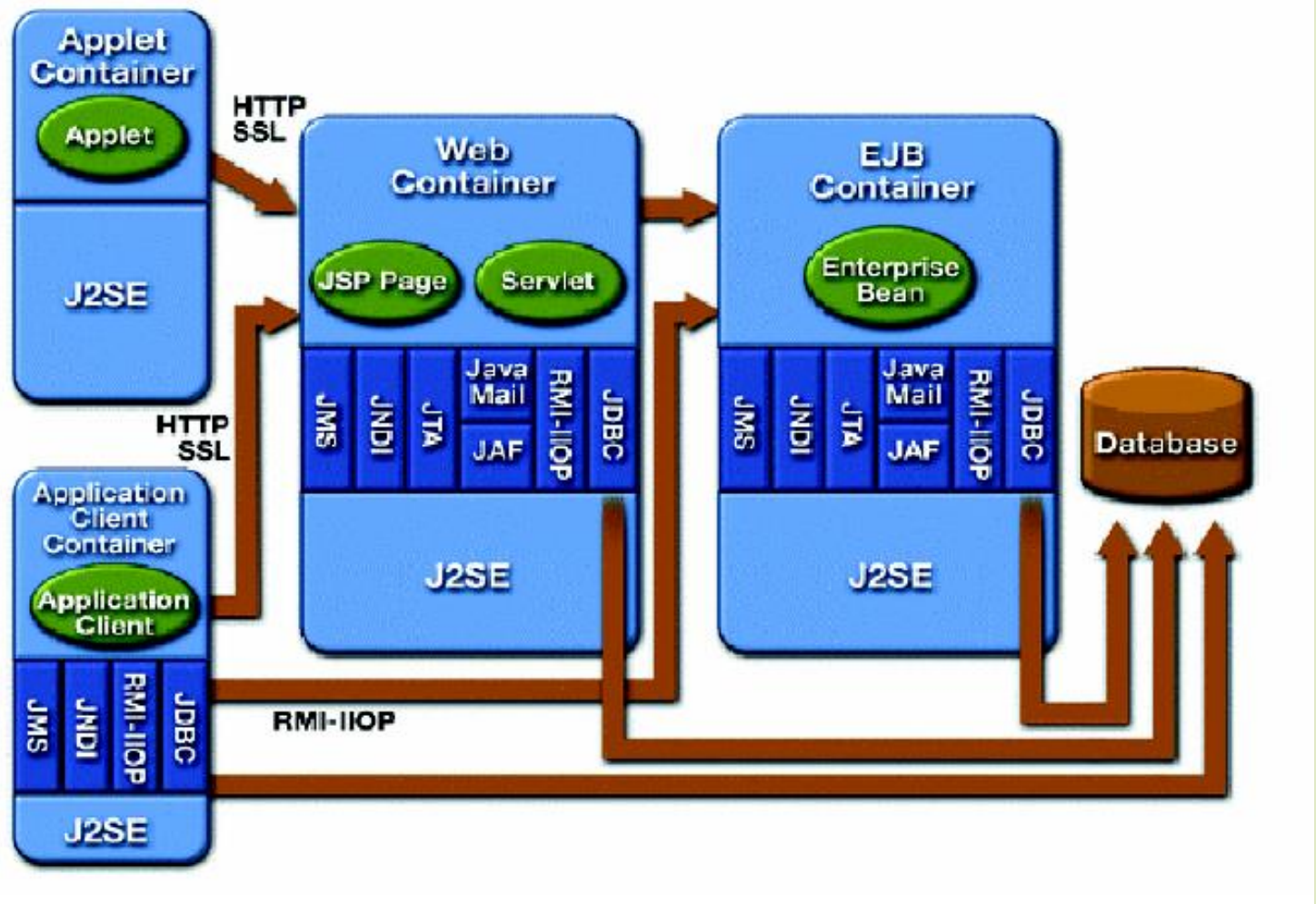
Java Enterprise Edition – JDBC

Desarrollo de Aplicaciones Cliente-Servidor

Dr. Jorge E. Villaverde

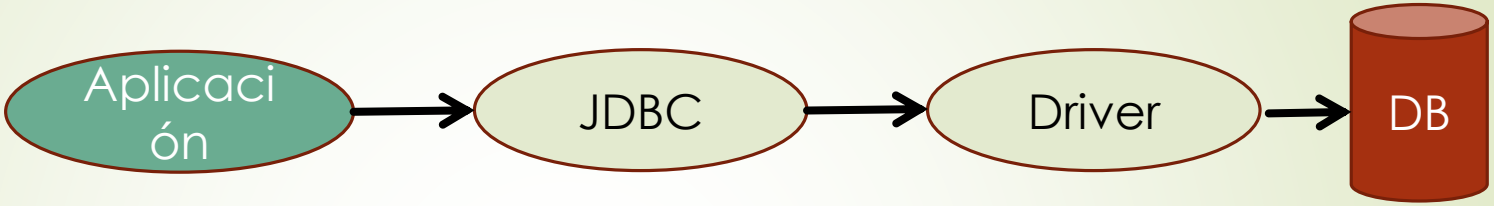
Contenedores



Java Data Base Connectivity

- JDBC Provee las librerías estándar para el acceso a BBDD Relacionales.
  - JDBC API Establece
    - La forma de conectarse a un BBDD
    - La forma de iniciar consultas
    - La forma de iniciar consultas almacenadas
    - La estructura de datos de los resultados de los queries
  - JDBC API ***NO Estable***
    - Sintaxis SQL
    - JDBC No es SQL Embebido
  - Incluido en el paquete **java.sql**

Arquitectura de JDBC

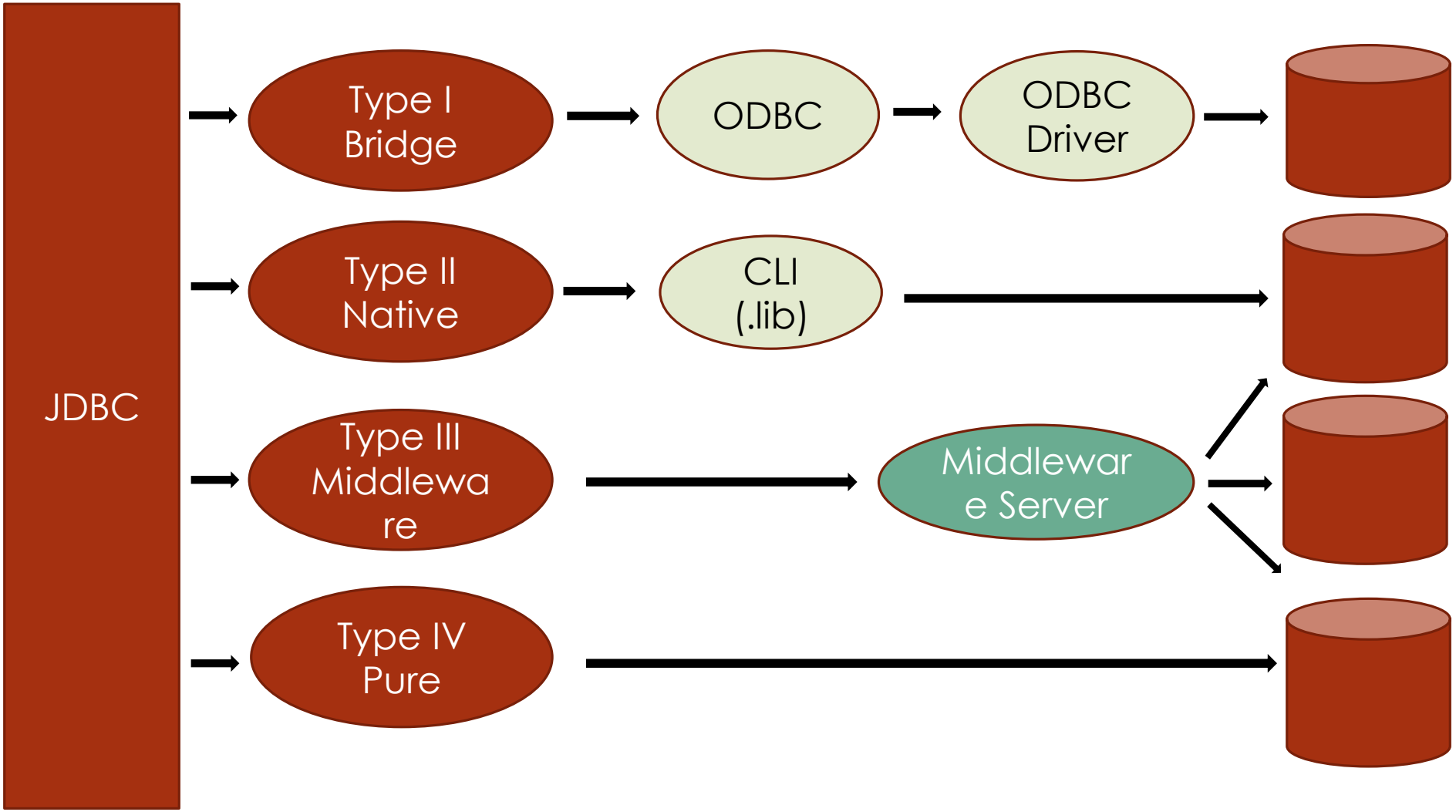


- Aplicación se comunica con la librería JBDC
- JDBC llama al driver de la BBDD
- El driver llama a una BBDD particular
- Se pude tener más de un Driver
- Se puede cambiar la BBDD *sin cambiar* el código de la aplicación.

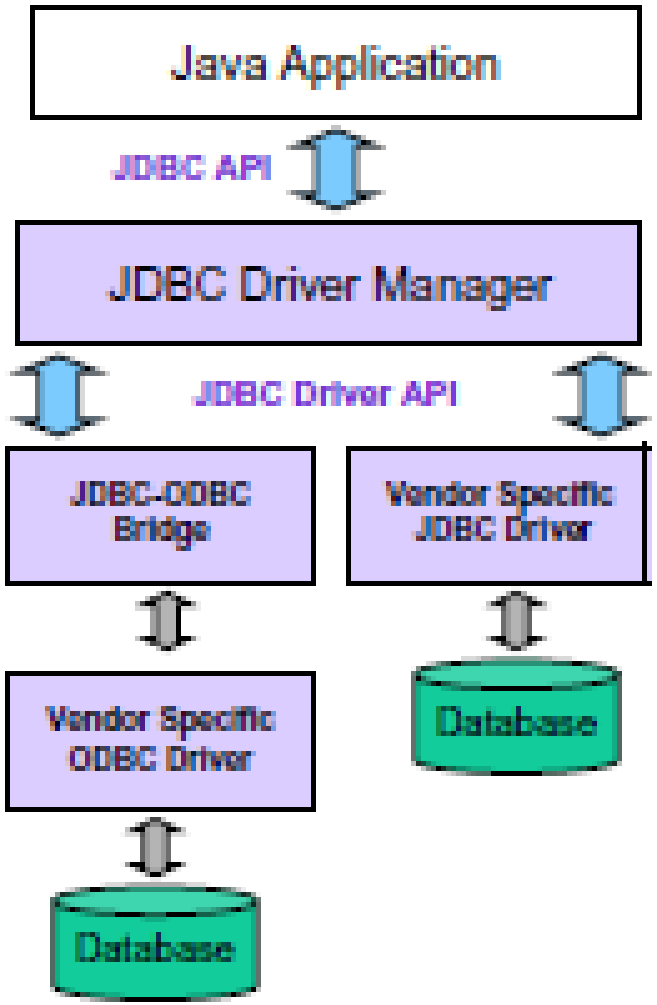
JDBC Drivers

- Type I: “Bridge”
- Type II: “Native”
- Type III: “Middleware”
- Type IV: “Pure”

JDBC Drivers



JDBC Drivers



- JDBC es una API Pura en Java
- JDBC Driver Manager es el que se comunica con los drivers específicos de los vendedores.
  - No se necesitan cambios del lado del servidor
  - Se utilizan drivers específicos en el cliente.

7 Pasos Básicos en JDBC

1. Cargar el Driver
2. Definir una URL de Conexión
3. Establecer la Conexión
4. Crear un objeto Statement
5. Ejecutar el Query
6. Procesar los resultados
7. Cerrar la Conexión

Ejemplo de 7 Pasos

1 Cargar el Driver

```
try {
    Class.forName("com.mysql.jdbc.Driver");
} catch (ClassNotFoundException e) {
   System.err.println("No existe el Driver para MySQL")
}
```

2 Definir la URL de Conexión

String url = "**jdbc:mysql://localhost/db1**";

Ejemplo de 7 Pasos

3 Establecer la Conexión

```
try {
  con = DriverManager.getConnection(url, user, password);
} catch (SQLException e) {
  System.err.println("Error al obtener una conexión");
}
```

4 Crear un objeto Statement

```
try {
  Statement stm = con.createStatement();
} catch (SQLException e) {
  System.err.println("Error al crear un Statement");
}
```

Ejemplo de 7 Pasos

5 Ejecutar el Query

```
try {
  String sql = "SELECT id, nombre FROM provincia";
  ResultSet rs = stm.executeQuery(sql);
} catch (SQLException e) {
  System.err.println("Error al ejecutar la consulta");
}
```

6 Procesar los resultados

```
try {
  while(rs.next()){
    int id = rs.getInt("id");
    String nombre = rs.getString(2);
    System.out.println("Provincia: id="+id+",
nombre="+nombre);
  }
} catch (SQLException e) {
  System.err.println("Error al crear un Statement");
```

Ejemplo de 7 Pasos

7 Cerrar la Conexión

```
try {
…
} catch (SQLException e) {
  System.err.println("Error al ejecutar la consulta");
} finally{
   if(rs != null)
      try {
         rs.close();
      } catch (SQLException e1) {}
   if(con != null)
      try {
         con.close();
      } catch (SQLException e) {}
}
```

Connection

- Representa una sesión con una BBDD específica.
- Dentro del contexto de una Connection, sentencias SQL se ejecutan y los resultados son devueltos.
- Se pueden tener multiples conexiones con una BBDD
  - Nota algunos Drives no soportan esto
- También proveen "metadata" – información sobre la estructura de los datos.
- También proveen métodos para tratar transacciones

Connection Methods

`Statement createStatement()`

- returns a new Statement object

`PreparedStatement prepareStatement(String sql)`

- returns a new PreparedStatement object

`CallableStatement prepareCall(String sql)`

- returns a new CallableStatement object

- ¿Por qué todos estos tipo diferentes de Statements? Optimización.

Statement

- Un ***Statement*** es un objeto utilizado para para ejecutar un SQL estático y obtener resultados producto de la consulta.

Statement Methods

`ResultSet executeQuery(String)`

- Execute a SQL statement that returns a single ResultSet.

`int executeUpdate(String)`

- Execute a SQL INSERT, UPDATE or DELETE statement. Returns the number of rows changed.

`boolean execute(String)`

- Execute a SQL statement that may return multiple results.

- ¿Por qué todos estos tipo diferentes de queries? Optimización.

ResultSet

- Un ResultSet provee acceso a una tabla de datos generada por la ejecución de un Statement.
- Sólo un ResultSet por Statement puede ser abierto.
- Las filas de la tabla son accedidas en forma secuencial.
- Un ResultSet mantiene un cursor apuntando a la fila actual de los datos.
- El método 'next' mueve el cursosr a la próxima fila.
  - No se puede volver a la fila anterior

ResultSet Methods

- boolean next()
  - Mueve el cursor al próximo registro
  - La primer llamada a next() activa la primer fila
  - Devuelve false si no hay más filas
- void close()
  - Libera el ResultSet
  - Permite reutilizar el Statement que generó el ResultSet
  - Es llamado automático por muchos de los métodos de Statement

ResultSet Methods

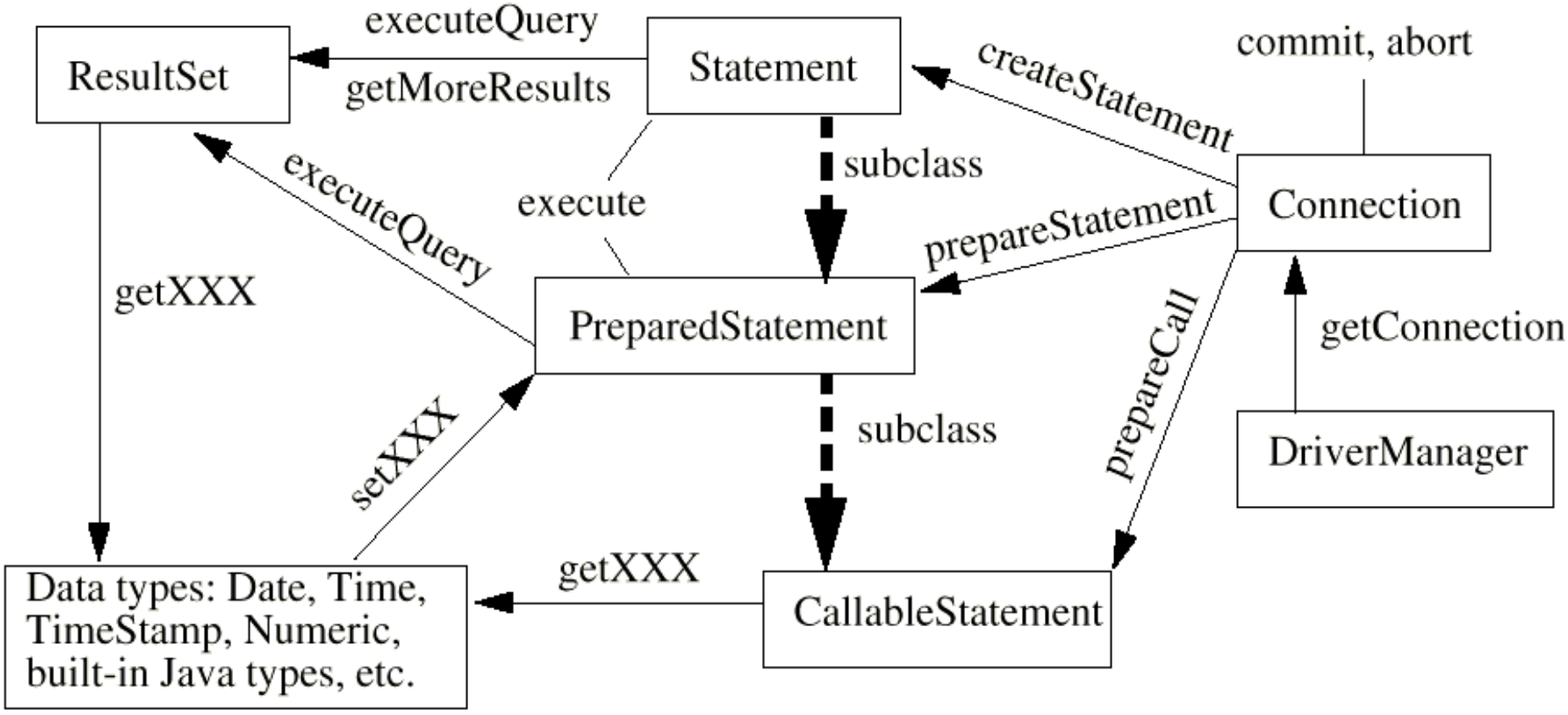
- String getString(String columnName)
- boolean getBoolean(String columnName)
- byte getByte(String columnName)
- short getShort(String columnName)
- int getInt(String columnName)
- long getLong(String columnName)
- float getFloat(String columnName)
- double getDouble(String columnName)
- Date getDate(String columnName)
- Time getTime(String columnName)
- Timestamp getTimestamp(String columnName)

isNull

- En SQL, NULL significa que el campo es vacio
- No es lo mismo que 0 o “”
- En JDBC, se debe perguntar explícitamente si un campo es null, llamando a

ResultSet.isNull(column)

Diagrama de JDBC



JDBC 2.0

- Scrollable result set
- Batch updates
- Advanced data types
  - Blobs, objects, structured types
- Rowsets
  - Persistent JavaBeans
- JNDI
- Connection Pooling
- Distributed transactions via JTS

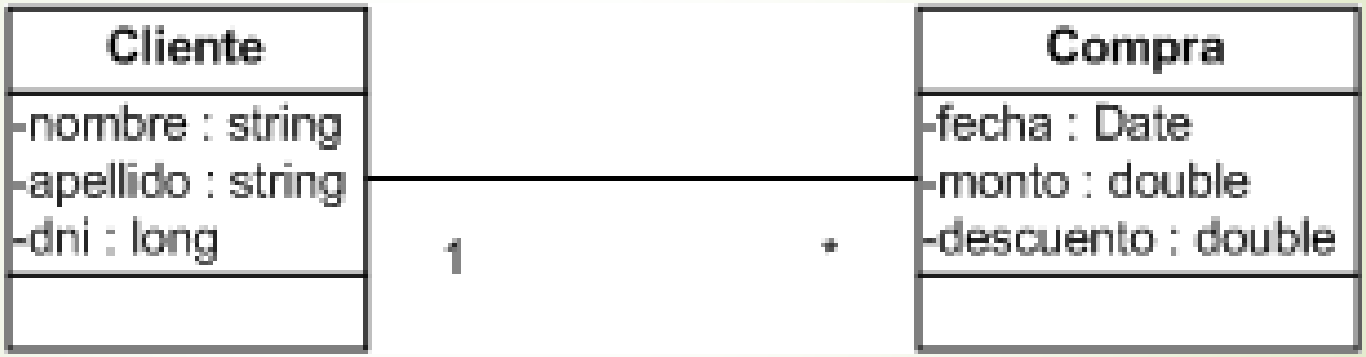
JDBC 3.0

- Metadata APIs
- Named parameters in CallableStatements
- Changes to data types
- Retrieving auto-generated keys
- Connector relationship
- ResultSet holdability
- Returning multiple results
- Connection pooling
- Prepared statement pooling
- Using savepoints in your transactions

Java Persistece API (JPA)

- API de persistencia desarrollada para la plataforma Java EE
- JPA fue originada a partir del trabajo del JSR 220.
- Ha sido incluida en el estándar EJB3
  - La API en sí misma, definida en javax.persistence.package
  - La Java Persistence Query Language (JPQL)
  - Metadatos objeto/relacional

Ejemplo de JPA



Preguntas